UNIVERSIDAD POLITÉCNICA DE CARTAGENA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

Estudio e Implementación de Mecanismos de Calidad de Servicio sobre una Arquitectura de Servicios Diferenciados

Autor
Ricardo Alarcón Llamas

Directora
Maria Dolores Cano Baños

Titulación
Ingeniería Técnica de Telecomunicación, especialidad Telemática

Cartagena, Enero 2003

Índice

Índice Figuras

Índice Tablas

Capítulo 1
Introducción

1.1 Introducción

En un principio, la mayor parte de las aplicaciones de Internet que ofrecían tráfico a la red eran servicios web, de acceso remoto, de correo electrónico o de transmisión de ficheros, que no tenían requerimientos específicos en cuanto a caudal mínimo, pérdidas de paquetes, retardos o varianza del retardo. Así, mediante el uso de una única clase de servicio, denominada Best Effort, se trataba por igual todo el tráfico generado por las aplicaciones.

Sin embargo, con el crecimiento de Internet y el éxito comercial que ha tenido, ha crecido exponencialmente el número de aplicaciones que introducen datos en la red, así como la necesidad de dar un tratamiento específico a cada tipo de tráfico. De este modo, irá apareciendo el término Calidad de Servicio (*QoS*, *Quality of Service*) que hace referencia a la capacidad de una red para proporcionar los mejores servicios al tráfico que vuelcan las aplicaciones en ella. Para gestionar la multitud de nuevas aplicaciones tales como video sobre *IP*, voz sobre *IP*, comercio electrónico y otras aplicaciones de tiempo real, las redes necesitan proporcionar Calidad de Servicio además del servicio best effort. Las diferentes aplicaciones tienen diversas necesidades de retardo, varianza del retardo, ancho de banda, pérdidas de paquetes y disponibilidad. Estos parámetros forman la base de la Calidad de Servicio. Por lo tanto, las redes *IP* actuales se deben de diseñar para solventar los requisitos de *QoS* a las aplicaciones. Por ejemplo, aplicaciones de voz sobre *IP* necesitarán un retardo muy bajo y un ancho de banda relativamente pequeño, mientras que la transmisión de ficheros requerirá más ancho de banda sin importar demasiado el retardo. Todas las redes se pueden beneficiar de los aspectos de Calidad de Servicio, redes de grandes, medianas y pequeñas empresas, proveedores de servicios de Internet, etc. Cada una de ellas tendrá unas necesidades de Calidad de Servicio distintas.

Una manera de hacer frente a estas necesidades es sobredimensionando las redes, es decir, empleando redes con una capacidad muy superior a las necesidades reales actuales. Pero esta forma de actuar tiene el problema del coste y desaprovechamiento de la tecnología (compras equipos que no empleas al 100%) además de que el rápido aumento del volumen de tráfico de la red actualmente puede dejar los sistemas sobredimensionados desbordados en muy poco tiempo.

Otra forma de solventar los nuevos requerimientos de Calidad de Servicio de las aplicaciones actuales será gestionando los recursos disponibles (el ancho de banda) de una forma eficiente. Para ello primero se deberán separar los distintos tipos de tráfico que transitan por la red para proporcionarles un tratamiento específico dependiendo de sus requerimientos. Para proporcionar una solución real a las necesidades de Calidad de Servicio extremo a extremo en las redes *IP* el *Internet Engineering Task Force* (*IETF*) define dos modelos: el modelo de Servicios Integrados (*IntServ*) y el modelo de los Servicios Diferenciados (*DiffServ*). *IntServ* sigue un modelo de Calidad de Servicio basado en señalización (*signaled-QoS*) donde los *host* informan de sus necesidades de *QoS* a la red, mientras que *DiffServ* trabaja sobre un modelo de *QoS* planificada donde los elementos de la red están configurados para servir múltiples clases de tráfico con diversos requerimientos de *QoS*. *IntServ* tiene serios problemas de escalabilidad debido a que los elementos de la red deben guardar estado de las diferentes conexiones. *DiffServ* soluciona este problema ya que serán los mismos paquetes de datos los que lleven información sobre el tratamiento particular que un nodo aplicará al paquete, librando así al nodo de mantener estado sobre los diferentes flujos de tráfico.

Las clases de Servicios Diferenciados son capaces de proporcionar un tratamiento individualizado a cada tipo de tráfico. Para ello la arquitectura de Servicios Diferenciados define una serie de elementos que realizan diferentes funciones sobre el tráfico de la red: acondicionarlo, clasificarlo, marcarlo, encolarlo, espaciarlo, etc.

Debido a la relativa novedad de este tipo de arquitectura y a las amplias posibilidades de configuración resulta interesante poder disponer de un estudio que englobe tanto una explicación clara y concisa de qué son los Servicios Diferenciados, de cómo configurar este tipo de servicios en redes reales, así como resultados experimentales de dichas configuraciones.

Por todo esto, el objetivo de este proyecto es el de explicar la arquitectura de Servicios Diferenciados con todos los elementos que la componen y ver cómo se implementa esa arquitectura en las redes actuales. Para ello se lleva a cabo un estudio sobre las diferentes herramientas que emplea el software de una de las multinacionales en el sector de tecnologías de la información más importantes (Cisco Systems) para implementar los distintos elementos que conforman esta arquitectura. Aunque como se ha visto, los Servicios Diferenciados, tal y como están definidos, pretenden ser una solución para proporcionar un tratamiento diferenciado al tráfico generado por las diferentes aplicaciones. En este proyecto se estudiarán también las posibilidades que ofrecen los elementos de la arquitectura de *DiffServ* a la hora de garantizar contratos. Por eso se realizarán varias pruebas para comprobar si, mediante el uso de los Servicios Diferenciados, los proveedores de servicios pueden garantizar contratos a sus clientes de la forma más justa posible.

Este proyecto se ha organizado de la siguiente manera: en el Capítulo 2 se hace una breve introducción de conceptos básicos cómo la Calidad de Servicio, los elementos con los que se mide y cómo influyen los dispositivos de la red en la Calidad de Servicio percibida por el usuario. También se hará un recorrido por las diferentes aproximaciones que se han realizado a lo largo de la historia de Internet hasta llegar a los Servicios Diferenciados. Se explicará en detalle la arquitectura de Servicios Diferenciados definida en el *RFC* 2475, analizando uno por uno los diferentes elementos que la componen. Por otro lado, se analizarán las diferentes disciplinas de gestión y control activo de la congestión y los algoritmos de token bucket empleados para las funciones de acondicionamiento del tráfico. Finalmente, se verán los requisitos y los problemas de implementar la arquitectura de Servicios Diferenciados en las redes actuales y cómo los fabricantes como Cisco lo solucionan.

En el Capítulo 3 titulado “Los Servicios Diferenciados en Cisco” se pretende mostrar el conjunto de herramientas con las que Cisco implementa la arquitectura de Servicios Diferenciados. Este capítulo consta de dos grandes bloques. En el primero de ellos se hace una introducción al software de red Cisco *IOS* que los routers de Cisco emplean para su configuración y funcionamiento. En esta introducción se explicarán conceptos básicos para poder utilizar este software de red, como son: los diferentes modos de comando del *router*, comandos de ayuda, la forma de actualizar ese software, etc. En el segundo gran bloque, se pasa a exponer las diferentes herramientas de las que el software de red de Cisco dispone para la implementación de los Servicios Diferenciados. Para ello, primero se hará un breve recordatorio de los diferentes elementos de la arquitectura viendo qué herramienta emplea Cisco para implementar cada uno de ellos y luego se explicarán en detalle cada una de esas herramientas.

En el Capítulo 4 de título "Configuraciones Experimentales’ se expondrán las configuraciones llevadas a cabo para probar las herramientas de *DiffServ* de Cisco y se comentarán los resultados experimentales obtenidos en dichas pruebas. Con estas pruebas se pretende mostrar las posibilidades que ofrecen los Servicios Diferenciados a la hora de garantizar contratos de una manera justa. Se analizará la influencia de los diferentes mecanismos de gestión y manejo de colas en dichos contratos y en reparto del ancho de banda en exceso de los enlaces. Así, en primer lugar se definirá el entorno de realización de las pruebas y las configuraciones necesarias para crear ese entorno: configuraciones iniciales de los routers, configuración de interfaces,

interconexiones, instalación etc. En segundo lugar se explicará como configurar el *router* para activar las distintas herramientas de *DiffServ* de Cisco, se presentarán los resultados obtenidos y se comentarán dichos resultados. Finalmente, se plantean las conclusiones a las pruebas llevadas a cabo.

Capítulo 2
Desarrollo Teórico

2.1 Calidad de Servicio *QoS*

Internet está cambiando muchos aspectos de nuestra vida: los negocios, el entretenimiento, la educación, etc. En el mundo empresarial es cada vez más usual el uso de *Internet* y de los servicios que proporciona (correo, páginas *web*, transferencia de ficheros) para crear *intranets* y *extranets,* con el fin de mejorar el funcionamiento de la actividad empresarial, así como de proporcionar un mejor servicio a sus clientes. Asimismo está creciendo el número de pequeñas y medianas empresas que se dedican a crear aplicaciones que funcionen bajo la red, ya sea para su propio beneficio o por encargo a otras empresas.

Desde un punto de vista empresarial, es esencial asegurar que dichas redes proporcionen los recursos que necesitan las aplicaciones utilizadas, adaptándose en cada momento a sus requisitos. Por ejemplo: La voz sobre *IP* requiere un retardo muy pequeño (del orden de 100 milisegundos) y un ancho de banda relativamente no muy grande y que oscila entre 8 Kbps y 64 Kbps, dependiendo del tipo de codificación usada. Otro ejemplo puede ser una aplicación de transmisión de ficheros, basada en el protocolo *FTP*, en donde no es tan importante el retardo, pero si la pérdida de paquetes, que puede ser muy perjudicial de cara a su rendimiento.

Cuando se habla de Calidad de Servicio (*Quality of Service, QoS*) [1], se está haciendo referencia a la capacidad de una red para proporcionar mejores servicios al tráfico que vuelcan las aplicaciones a la red y que funcionan bajo distintas tecnologías incluyendo: *Frame Relay*, *ATM (Asynchronous Transfer Mode)*, *Ethernet* y 802.1, *SONET*, etc. En particular los servicios de los que hablamos son:

- Soportar ancho de banda dedicado
- Reducir la pérdida de paquetes
- Evitar y gestionar las congestiones
- Espaciar el tráfico de una red (*Shaping)*
- Fijar prioridades del tráfico a través de la red

La *QoS* extremo a extremo (*end-to-end*) es la habilidad de una red para proporcionar al tráfico el servicio que necesita desde un extremo de una red hasta otro. Según el nivel de *QoS,* la *Internet Engineering Task Force (IETF)* define tres tipos de modelos de servicio (también llamados niveles de servicio) que son: *Best Effort*, Servicios Integrados (*Integrated Services, IntServ*) y Servicios Diferenciados (*Differenciated Services, DiffServ*).

A la hora de decidir qué tipo de servicio emplear en una red hay que considerar los siguientes factores:

- El problema que se está intentando resolver. Cada uno de los tres servicios es apropiado para un cierto uso.
- El tipo de capacidad que se quiere asignar a los recursos.

- La relación coste-beneficio. Por ejemplo, el coste de implementar un servicio diferenciado es mayor que el coste de un servicio B*est Effort*.

Los siguientes tres componentes son necesarios para proporcionar *QoS* a través de una red heterogénea:

- *QoS* en un elemento de la red, que incluya: herramientas para formación de colas (*queueing)*, planificación *(scheduling)* y espaciado del tráfico *(traffic shaping)*.

- Técnicas de señalización de *QoS,* para la coordinación de la *QoS* de extremo a extremo que ocurre entre los elementos de la red.

- Funciones de policía y gestión *(policing & management)* de *QoS* para controlar y administrar el tráfico de extremo a extremo a través de la red.

A continuación, se expondrá brevemente en qué consiste cada uno de los anteriores servicios y se seguirá con la evolución que se ha seguido hasta la aparición de los Servicios Diferenciados tomando como referencia las ventajas que ofrece sobre los restantes servicios.

2.1.1 Elementos para medir la *QoS* percibida

Para estudiar cómo influyen los elementos de la red, nodos y enlaces entre nodos, en la *QoS* percibida por los usuarios se tendrá que ver como influyen esos elementos en los atributos que conforman la *QoS* que son [2]:

- El caudal (*Throughput*)

- El retardo (*Delay*)

- La varianza del retardo (*Jitter*)

- Las pérdidas (*Loss*)

2.1.1.1 Throughput

El caudal o *throuhgput* es un término genérico que describe la capacidad de un sistema para transferir datos. En redes *TCP/IP* el *throughput* se define y se mide de varias formas:

- La tasa de *bytes* o paquetes que va por el circuito,

- la tasa de *bytes* o paquetes del flujo de una aplicación específica,

- la tasa de *bytes* o paquetes del conjunto de flujos de un nodo a otro o

- la tasa de *bytes* o paquetes del conjunto de flujos de una red a otra.

El parámetro más directo que un *router* puede configurar para controlar el *throughput* es la cantidad de ancho de banda reservado para los diferentes tipos de paquetes.

- En el servicio de *Best Effort* clásico el *router* no controla específicamente la cantidad de ancho de banda asignado a las diferentes clases de tráfico. Durante periodos de congestión los paquetes se colocan en una cola *FIFO First-in First-out* (primero en entrar primero en salir). Los datagramas *UDP* al no reducir su tasa de transmisión

cuando las colas se llenan se quedan con todo el ancho de banda del enlace, aumentando el *throughput* percibido por sus fuentes y reduciendo considerablemente el *throughput* de las fuentes *TCP*.

- Cuando se diferencia el tráfico en clases, se pueden crear varias colas para cada tipo de tráfico y controlar el ancho de banda reservado para cada uno de estos tráficos. De esta forma, en caso de congestión, los flujos de tráfico *UDP* no acapararán todo el ancho de banda del enlace y se podrá repartir el *throughput* entre los diferentes flujos de tráfico.

2.1.1.2 Delay

Retardo o latencia (*delay*) es la cantidad de tiempo que lleva transmitir un paquete desde un punto de la red a otro. Hay varios factores que influyen en el retardo experimentado por un paquete que atraviesa la red:

- Retardo de enrutamiento (*forwarding delay*),
- retardo en colas (*queueing delay*),
- retardo de propagación (*propagation delay*),
- retardo de serialización (*seralization delay*).

La siguiente figura muestra que el retardo de extremo a extremo se puede calcular como la suma de los retardos individuales de encaminamiento, colas, serialización y propagación que se producen en cada nodo y enlace en la red.

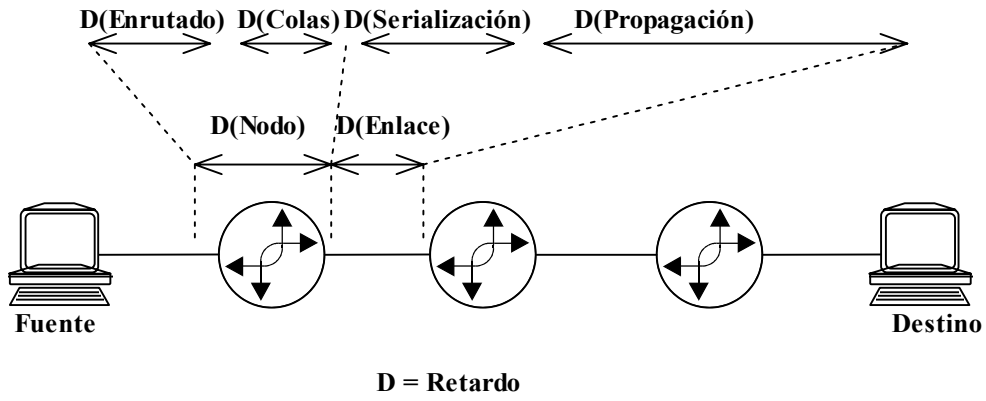

Figura 2. 1: Retardo de Extremo a Extremo

Fuentes de retardo en la red:

- **Retardo de enrutamiento:** Es la cantidad de tiempo que tarda un *router* en recibir un paquete, tomar una decisión de encaminamiento y transmitir el paquete a través de un puerto de salida no congestionado. Se mide normalmente en decenas o cientos de microsegundos.
- **Retardo de colas:** Es la cantidad de tiempo que espera un paquete en una cola hasta que se sirve. Durante periodos de congestión se puede controlar el retardo en las colas mediante las disciplinas de gestión de memorias y de servicio de colas.

- **Retardo de propagación:** Es la cantidad de tiempo que tardan los electrones o fotones en atravesar un enlace físico. Se basa en la velocidad de la luz y se mide en milisegundos. Ya que no se puede cambiar la velocidad de la luz, en fibra óptica no se podrá controlar el retardo de propagación a menos que se reduzca la distancia del enlace.

- **Retardo de serialización:** Es la cantidad de tiempo que se tarda en colocar los bits de un paquete en el cable cuando el *router* transmite un paquete. Se mide en milisegundos y va en función del tamaño del paquete y de la velocidad del puerto. Debido a que no hay prácticamente ningún mecanismo para controlar el tamaño de los paquetes de la red (que no sea reducir el MTU o forzar la fragmentación del paquete), la única acción que se puede llevar a cabo para reducir el retardo de serialización es instalar interfaces de alta velocidad en el *router*.

Sin embargo, es importante recordar al hablar del retardo que el *router* representa sólo una parte del camino de extremo a extremo y que se deben considerar otros factores:

- El papel de los cuellos de botella dentro de *host* y servidores.
  - *Retardos de Planificación del Sistema Operativo*
  - Retardos de enmarcado de la capa física
  - Retardos de codificación *CODEC*, compresión y empaquetado.
- La calidad de las implementaciones de *TCP/IP* que se ejecutan en estos sistemas
- La estabilidad del enrutado en la red.

2.1.1.3 Jitter

Jitter es la variación del retardo en el tiempo entre paquetes consecutivos que forman parte del mismo flujo (ver Figura 2. 2). Se puede medir el *jitter* usando diferentes técnicas, incluyendo la media, desviación típica, máximo o mínimo del tiempo de llegada entre los paquetes, para paquetes consecutivos de un mismo flujo.

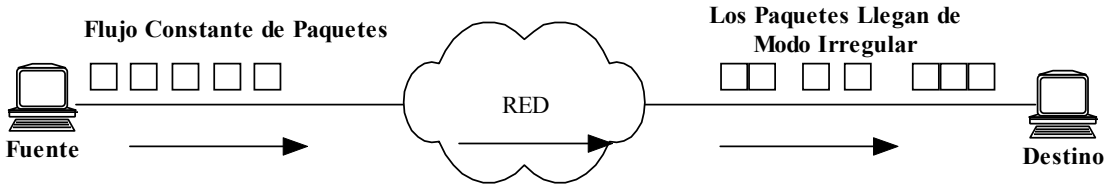

Figura 2. 2: Varianza del Retardo o *Jitter*

La principal fuente de *jitter* es la variación del retardo en las colas para paquetes consecutivos de un mismo flujo. Otra fuente potencial de *jitter* es que los paquetes consecutivos de un mismo flujo sigan caminos físicos diferentes. Además, el *jitter* crece exponencialmente con el aumento de la utilización del ancho de banda al igual que el retardo. Por todo ello, el *jitter* influye en la *QoS* percibida, sobre todo en aplicaciones de voz o vídeo.

2.1.1.4 Loss

Hay tres fuentes de pérdidas de paquetes en una red *IP*, como se ve en la Figura 2. 3:

- Una rotura en un enlace físico que evita la transmisión de un paquete,
- un paquete corrupto debido al ruido detectado por un sistema de *checksum* y
- desbordamiento de las memorias producidas por la congestión de la red.

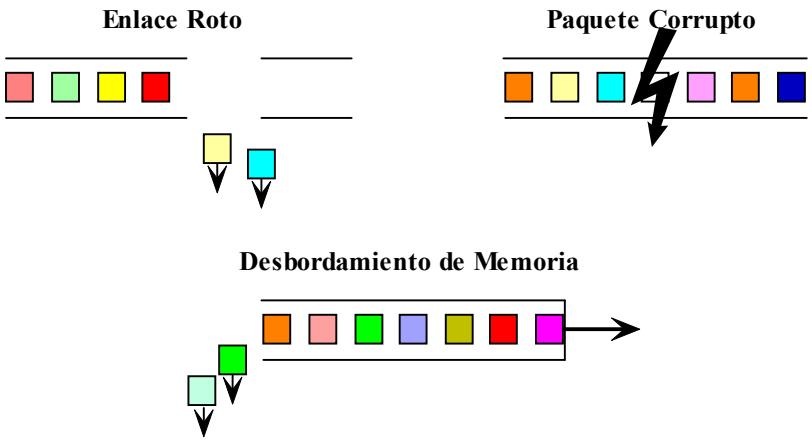

Figura 2. 3: Posibles Fuentes de Pérdidas

Nótese que no todas las pérdidas de paquetes son malas. Por ejemplo, el *router* puede realizar el descarte selectivo de algunos paquetes para así evitar la congestión.

2.1.1 Evolución hasta los Servicios Diferenciados en las redes *IP*

La intención de proporcionar un servicio para las redes *IP* mejor que la clase de servicio *Best Effort* tiene más de 20 años. En los apartados siguientes se verá la evolución desde la clase *Best Effort* que no ofrecía ningún tipo de *QoS* hasta las clases de Servicios Diferenciados que son capaces de proporcionar un tratamiento individualizado a cada tipo de tráfico [2].

2.1.1.5 Best effort

IP tradicionalmente ofrecía sólo una clase de servicio, conocida como *Best Effort*, donde todos los paquetes que atravesaban el *router* se trataban con la misma prioridad. *Best Effort* significa que *IP* hace un esfuerzo razonable para hacer llegar cada datagrama incorrupto a su destino, pero no garantiza que un paquete no llegue corrupto, duplicado o reordenado. Además no garantiza nada sobre el caudal, retardo, varianza del retardo o pérdidas que experimentará un paquete.

Así, el servicio *Best Effort* sin el soporte proporcionado por los protocolos de transporte inteligentes como *TCP* puede llevar al caos. La única razón de que el servicio de *Best Effort* funcione en las redes globales *IP* se debe a que *TCP* no compromete a la red cuando experimenta congestión.

En este modelo las aplicaciones mandan los datos cuando pueden, en cualquier cantidad, y sin pedir permiso ni informar a la red. La red entrega los datos si puede, por lo que no proporciona confiabilidad. Una forma de implementar este servicio es mediante colas *FIFO* en los nodos.

2.1.1.6 La primera aproximación: *RFC* 791

En septiembre de 1981, la *Request For Comments RFC* 791 [3] estandariza el protocolo *IP* y reserva el segundo *byte* de la cabecera *IP* como el campo de tipo de servicio (Type of Service, *ToS*). Los bits del *byte ToS* se definen en el *RFC* 1349 como muestra la Figura 2. 4:

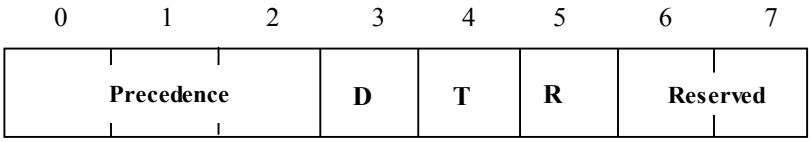

Figura 2. 4: byte Type of Service IPv4 (ToS)

Un nodo puede poner los tres primeros bits en el *byte ToS* (bits de precedencia) para seleccionar la prioridad relativa o precedencia del paquete. Los tres bits siguientes especifican un retardo (D) normal o bajo, un caudal (T) normal o alto y una confiabilidad (R) normal o alta. Los dos bits finales del *byte ToS* se reservan para uso futuro. Sin embargo, muy pocas arquitecturas ofrecen clases de Servicios Diferenciados en las redes *IP* usando estos bits.

En caso de que haya bloqueo en un nodo los paquetes con el valor más bajo en el campo *Precedence* pueden ser descartados. Además cada paquete puede ser marcado para recibir un tipo de servicio según el campo "tipo de servicio definido" (*Defined Type of Service, DTS bits*). Este campo va del bit 3 al 5 del byte ToS y especifica: bajo retardo (*minimize delay*), alta confiabilidad (*maxime reliability*), etc…

En un principio parece que con este esquema se conseguiría resolver los problemas de escalabilidad en las redes puesto que no es necesario el intercambio de información entre los nodos para asegurar *QoS* ya que toda la información necesaria va en cada paquete de datos. No obstante, este esquema presenta las siguientes limitaciones:

- El esquema de *IP Precedence* sólo permite especificar la prioridad relativa de los paquetes, es decir, en caso de que haya congestión y los paquetes tengan el mismo valor de *Precedence* no permite especificar diferentes prioridades a la hora de descartarlos. Por ejemplo, un administrador de una red puede querer que tanto los paquetes de tráfico *HTTP* como los de *Telnet* lleven el mismo valor en el campo *Precedence* y que cuando haya congestión, los paquetes de *Telnet* se descarten antes que los de *HTTP*. Esto no es posible hacerlo con el esquema de *IP-Precedence*.

- Los tres bits del campo *IP Precedence* restringen el número de posibles tipos de precedencia a ocho. Además, las clases de más alta prioridad *Network Control* e *Internet Network Control* se reservan normalmente para los mensajes enviados entre *routers* que son mensajes para actualizar tablas de encaminamiento, mensajes *ICMP* para informar sobre su estado, etc. Aunque estos paquetes son necesarios para mantener el buen estado de la red reducen el número de clases para el tráfico generado por usuarios a 6.

- Ni el *IP Precedence* ni los bits *DTS* son implementados correctamente por los vendedores de redes hoy en día.

Todo lo anterior reduce las oportunidades de implementar con éxito *QoS* extremo a extremo usando este esquema.

2.1.1.7 La segunda aproximación: El Modelo de Servicios Integrados (*IntServ*)

Hacia 1993, el objetivo era que servicios de tiempo real compartieran la red *IP* de modo simultáneo con los servicios tradicionales (sin requerimientos de tiempo real). El resultado es la creación de la arquitectura de Servicios Integrados.

Es un modelo de servicio múltiple que puede albergar muchos requerimientos de *QoS.* En este modelo la aplicación envía un mensaje de señalización a la red para solicitar un tipo de servicio que le proporcione el ancho de banda y el retardo máximo aceptable para los datos a enviar. La aplicación envía solo los datos en el momento que recibe la confirmación por parte de la red.

Este modelo utiliza el protocolo de reserva de recursos *Resource Reservation Protocol* (*RSVP*) [12] empleado por las aplicaciones para especificar sus requerimientos de *QoS* a la red. La figura 2.5 muestra el funcionamiento del protocolo *RSVP*.

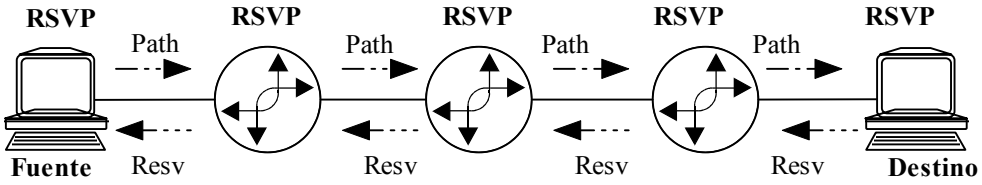

Figura 2. 5: Protocolo de Reserva de Recursos *RSVP*

Los inconvenientes de los Servicios Integrados son:

- Todos los nodos que forman la red, incluidos los sistemas de los extremos que pueden ser *Pc´*s o servidores, deben entender perfectamente el protocolo *RSVP*.

- Las reserva de recursos en cada uno de los *routers* es "suave", lo que significa que es necesario que se refresque periódicamente por lo que la reserva puede anularse si los paquetes de refresco se pierden. Hay mecanismos para evitar este problema pero lo hacen a costa de añadir más complejidad al protocolo *RSVP*.

- La necesidad de memoria para almacenar las reservas en los *routers* aumenta al aumentar el número de reservas.

- Dado el gran número de mensajes que se tienen que intercambiar los *routers* para mantener el estado de las reservas resulta poco escalable.

2.1.1.8 Servicios Diferenciados (*DiffServ*)

Hacia 1995, los proveedores de servicios y varias instituciones académicas comienzan a examinar aproximaciones alternativas mejores que una simple clase de servicio *Best Effort*, pero esta vez usando mecanismos que proporcionen escalabilidad. La figura 2.6 muestra el coste y la complejidad relativa de las soluciones vistas hasta ahora:

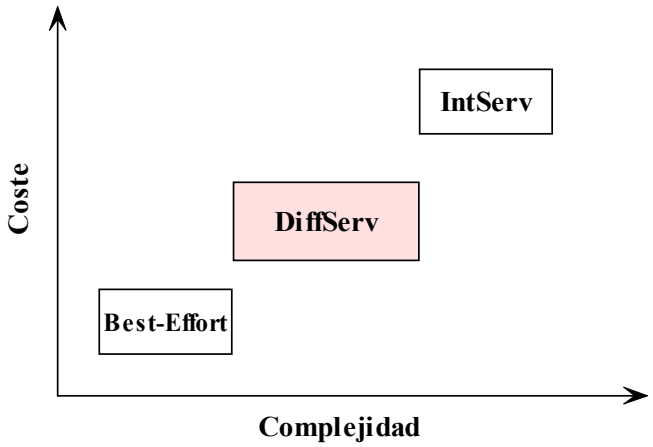

Figura 2. 6: Gráfica de Coste y Complejidad de las Diferentes Clases de Servicio de Internet.

La *IETF* acaba el *RFC* para *DiffServ* hacia finales de 1998. El objetivo que se marcó el grupo de trabajo encargado de su desarrollo fue: "*Hay una clara necesidad de un método completo y simple para proporcionar diferentes clases de servicio para el tráfico en Internet y que soporte diferentes tipos de aplicaciones y requerimientos del mundo de los negocios. El acercamiento del servicio diferenciado para proporcionar QoS en redes emplea una serie de componentes bien definidos con los cuales se pueden construir una gran variedad de arquitecturas. Se usa un grupo reducido de bits (en IPv4 el byte ToS y en IPv6 el byte Class) para marcar un paquete con el fin de que reciba un tratamiento particular de encaminamiento en cada nodo de la red. Es necesario definir una interpretación y uso común para este grupo de bits para asegurar la compatibilidad entre equipos de múltiples fabricantes y para su uso en los dominios de Internet. Así, el grupo de trabajo ha propuesto como estándar que el grupo de bits sea un campo de 6 bits llamado el campo DS (DiffServ). Los RFC´s 2474* [4] *y 2475* [5] *definen la arquitectura de DiffServ y el uso general del campo DS (reemplazando la definición del byte ToS de la RFC 1349* [6]*).*"

El grupo de trabajo de los Servicios Diferenciados (*DiffServ Working Group*) cambia el nombre del octeto *ToS* del *IPv4* por el *byte DS* [4] y define significados nuevos para cada uno de los bits (ver figura 2.7). Las nuevas especificaciones para el campo *DS* son las mismas en el octeto *ToS* del *IPv4* que en el octeto clase de tráfico del *IPv6*.

El grupo de trabajo de los Servicios Diferenciados del *IETF* divide el *byte DS* en dos subcampos:

- Los seis bits de mayor peso se conocen como el *Differentiated Services Codepoint* (*DSCP*). El *DSCP* lo emplea un *router* para determinar el tratamiento que recibirá un paquete en cada nodo a lo largo de un dominio de Servicios Diferenciados.

- Los dos bits de menor peso *Currently Unused* (*CU*) están reservados para uso futuro.

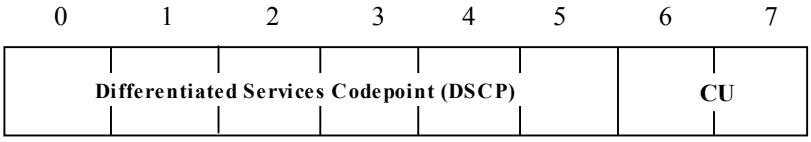

Figura 2. 7: Campo *DSCP* en *IPv4*

2.2 Arquitectura para los Servicios Diferenciados

La arquitectura de los Servicios Diferenciados [5][7][13] está basada en un modelo simple en el que el tráfico que entra a una red es clasificado, posiblemente acondicionado en los límites de la red y asignado a diferentes grupos de tráfico de idéntico comportamiento (*Behavior Aggregates, BA)*. Cada *behavior aggregate* se identifica con un único *DSCP*. En el interior de la red, los paquetes se encaminan de acuerdo a un tratamiento particular o *Per-Hop Behavior (PHB)* asociado con el *DS codepoint*.

2.2.1 Dominio de Servicios Diferenciados (*DiffServ Domain*)

Es un grupo continuo de nodos que implementan los Servicios Diferenciados *(DS nodes)*. Un dominio *DS (DS domain)* tiene muy bien definidos los límites, consistentes en los nodos frontera *(DS boundary nodes)*, que clasifican y posiblemente acondicionan el tráfico de entrada al dominio, asegurando que los paquetes que circulan por él vayan marcados con el apropiado *PHB* de uno de los grupos *PHB* implementados dentro del dominio. Los nodos interiores realizarán el encaminamiento de los paquetes basándose en su *DSCP* y así, según éste, los nodos tratarán el tráfico de una forma u otra.

La introducción de nodos que no implementen los Servicios Diferenciados *(non-DS-compliant nodes)* dentro de un dominio *DS* da resultados impredecibles y puede llegar a impedir la capacidad de satisfacer los contratos del cliente con su proveedor de servicios *(Service Level Agreements, SLAs)*.

Para entender la arquitectura usaremos la Figura 2. 8 y en las siguientes secciones explicaremos cada uno de los elementos que la componen.

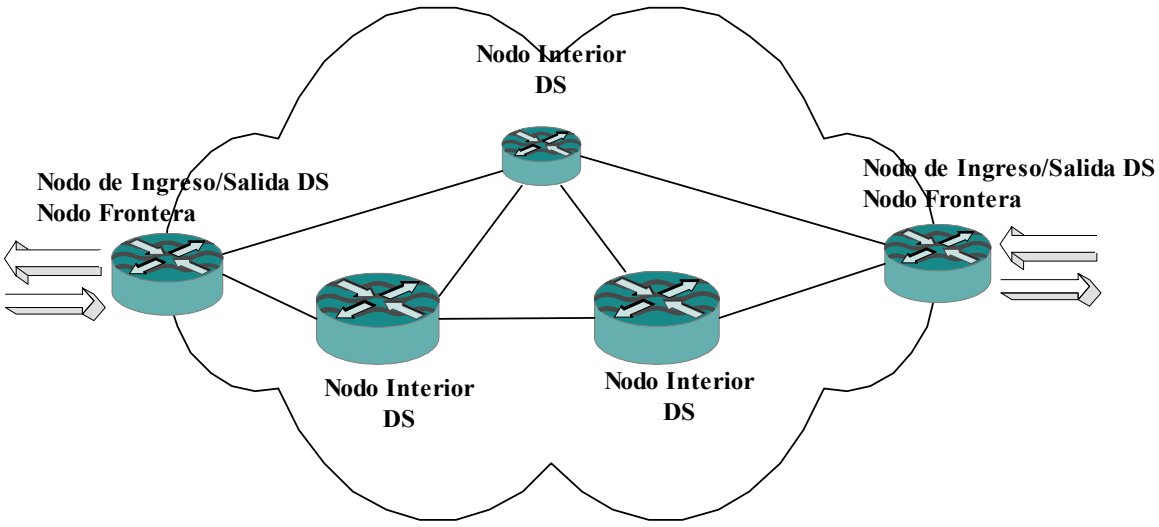

Figura 2. 8: Dominio de Servicios Diferenciados

2.2.1.1 Nodos Frontera DS (*DS Boundary Nodes*)

Los nodos frontera interconectan el dominio *DS* con otros dominios *DS* o con dominios no capacitados para soportar los Servicios Diferenciados *(non-DS-capable domains)*. Deben ser

capaces de aplicar el *PHB* apropiado a paquetes basados en el *DSCP*; si no se pueden producir resultados inesperados. Además puede ser necesario que apliquen funciones de acondicionamiento al tráfico definidas por un *Traffic Conditioning Agreement (TCA)* entre su dominio DS y los dominios DS conectados con ellos. Por otro lado, un *host* (ordenador) de una red puede actuar como un *boundary node* para el tráfico generado por sus aplicaciones y si no lo hará su nodo más cercano.

2.2.1.2 Nodos Interiores DS (*DS Interior Nodes*)

Los nodos interiores sólo están conectados con otros nodos *DS* interiores o frontera dentro del mismo dominio *DS*. Deben de ser capaces de aplicar el *PHB* apropiado a paquetes basados en el *DS codepoint*; si no se pueden producir de nuevo resultados inesperados. Los nodos interiores *DS* realizan funciones de acondicionamiento del tráfico más limitadas que los nodos *DS* frontera ya que si no serían prácticamente idénticos a ellos. Una de estas funciones puede ser el remarcado de los *DS codepoints*.

2.2.1.3 Nodos DS de Ingreso y de Salida (*DS Ingess Node and Egress Node*)

Los nodos *DS* frontera actúan a la vez como nodos de entrada y salida *(DS Ingress and Egress nodes)* para el tráfico de un dominio *DS*. Un nodo *DS* de entrada es el responsable de asegurar que el tráfico que entra en un dominio *DS* respeta algún *TCA* entre él y el otro dominio al que el nodo está conectado. Un nodo *DS* de salida debe realizar funciones que acondicionen el tráfico dirigido hacia un dominio directamente conectado con él, dependiendo también del *TCA* entre los dos dominios.

2.2.2 Región de Servicios Diferenciados (*DS Region*)

Es un conjunto de uno o más dominios *DS* contiguos. Es capaz de dar soporte para los Servicios Diferenciados en cualquier ruta que pertenezca a la región.

Los dominios *DS* dentro de una región podrán soportar diferentes grupos *PHB* internamente. Sin embargo, para permitir a los servicios expandirse a lo largo de los diferentes dominios, los dominios contiguos deberán establecer un *SLA* que defina (explícita o implícitamente) un *TCA* que especifique como transita el tráfico desde un dominio *DS* hacia otro.

Es posible que algunos dominios dentro de una región *DS* adopten unas políticas comunes, lo que eliminaría la necesidad de acondicionadores del tráfico entre estos dominios.

2.2.3 Clasificación y Acondicionado del Tráfico

El *SLA* puede especificar la clasificación de paquetes, las reglas de remarcado, los perfiles del tráfico y las acciones a llevar a cabo en los flujos de tráfico cuando éstos se acomodan o no a los perfiles dados. El *TCA* entre dominios deriva (explícita o implícitamente) de este *SLA*.

Las políticas de clasificación de paquetes identifican el subconjunto de tráfico que puede recibir un servicio diferenciado y lo condicionan y/o mapean a uno o más *behavior aggregates* (mediante el remarcado del *DS codepoint*) dentro de un dominio DS.

El acondicionado de tráfico *(Traffic Conditioning)* realiza mediciones *(metering)*, espaciado *(shaping)*, funciones policía *(policing)* y/o remarcado *(remarking)* para asegurar que el tráfico que entra a un dominio *DS* está conforme con las reglas especificadas en el *TCA*. La magnitud

del *Traffic Conditioning* requerido depende de las especificaciones del servicio ofrecido, y puede ir desde un simple remarcado del *DSCP* a complejas operaciones de *policing* y *shaping*.

2.2.3.1 Clasificadores (*Classifiers*)

Los clasificadores de paquetes seleccionan los paquetes dentro de un flujo de tráfico basándose en el contenido de alguna porción de la cabecera. Definimos dos tipos de clasificadores. Los *BA* (Behavior Aggregate Classifier) clasifican los paquetes basándose solamente en el *DS codepoint*. Los clasificadores multi-campo (*Multi-Field classifiers)* seleccionan paquetes basándose en el valor de la combinación de uno o más campos de la cabecera, tales como la dirección de origen, de destino, campo *DS*, ID de protocolo, número de puerto origen y destino y otra información tal como la interfaz de entrada.

Los clasificadores se emplean para realizar comprobaciones en los paquetes del cumplimiento de varias reglas para procesados posteriores. Los clasificadores deben configurar mediante algún procedimiento que esté en concordancia con el *TCA* apropiado. Además, los clasificadores deben autentificar la información que se usa para clasificar los paquetes.

2.2.3.2 Perfiles de Tráfico (*Traffic Profiles*)

Un Perfil de Tráfico *(Traffic Profile)* especifica las propiedades temporales de un flujo de tráfico seleccionado por un clasificador. Proporciona reglas para determinar cuando un paquete en particular está dentro del perfil *(in-profile)* o fuera de él *(out-of-profile)*. Estar dentro del perfil especificado significa que el paquete cumple todas las reglas de ese perfil en particular, es decir, que las propiedades del paquete coinciden con las especificadas por el perfil. Por ejemplo, un perfil basado en un algoritmo de *Token Bucket* puede verse como:

codepoint = X, usa token-bucket r, b

El perfil visto anteriormente indica que todos los paquetes marcados con el *DS codepoint* X se deberán medir mediante un medidor de *Token Bucket* con tasa r y tamaño de ráfaga b. En este ejemplo los paquetes que estén *out-of-profile* serán aquellos paquetes que lleguen cuando no haya suficientes *tokens* disponibles en el cubo. El concepto de *in-* y *out-of-profile* se puede extender a más de dos niveles.

Acciones diferentes de condicionado se les pueden aplicar a los paquetes *in-profile* y *out-of-profile*. A los paquetes *in-profile* se les puede permitir entrar al dominio *DS* sin más condiciones o, alternativamente, se les puede cambiar su *DS codepoint*. Los paquetes *out-of-profile* se pueden encolar hasta que estén *in-profile (shaped)*, descartados *(policed)* o marcados con un nuevo *codepoint (re-marked)*.

2.2.3.3 Acondicionadores de Tráfico (*Traffic Conditioners*)

Un Acondicionador de Tráfico *(Traffic Coditioner)* puede contener los siguientes elementos: un medidor *(Meter)*, un marcador *(Marker)*, un espaciador *(Shaper)* y un descartador *(Dropper)*. Un clasificador selecciona un flujo de tráfico y dirige los paquetes a un acondicionador del tráfico *Traffic Conditioner*. El medidor se usa para comparar el flujo de tráfico con algún perfil de tráfico. El resultado de la medida respecto a un paquete en particular (p.e: si el paquete está in- ó out-of-profile) puede afectar a las acciones de marcado, descarte o espaciado.

La figura 2.9 muestra el diagrama de bloques de un clasificador de tráfico y un acondicionador de tráfico. Notar que el acondicionador de tráfico no tiene porqué tener estos cuatro elementos (Meter, Marker, Shaper y Dropper).

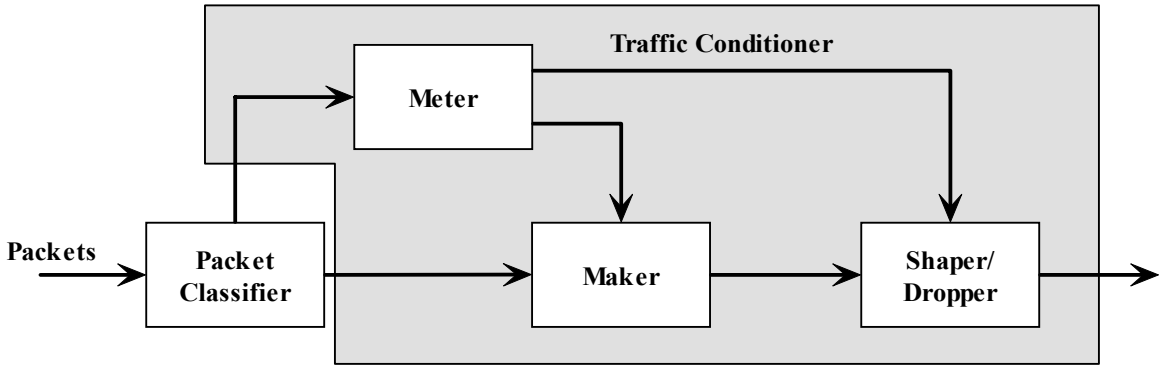

Figura 2. 9: Diagrama de Bloques de un Clasificador y de un Acondicionador de Tráfico

2.2.3.3.1 Medidores *(*Meters*)*

Los medidores de tráfico miden las propiedades temporales del flujo de paquetes seleccionado por un clasificador comparando esas propiedades con las de un perfil de tráfico *Traffic Profile* especificado en un *TCA*. Un medidor pasa información de estado a otras funciones de acondicionado (*Marker* or *Shaper*/*Dropper*) para activar una acción particular para cada paquete los cuales pueden estar *in-* o *out-of-profile*.

Por ejemplo, un paquete de *FTP* se clasifica en una determinada clase, las políticas aplicadas a esta clase especifican que para el tráfico de esa clase hay un ancho de banda del X%, superado éste los paquetes podrán ser descartados. Pues bien, el medidor será el encargado de comprobar si el tráfico de esa clase se ajusta a la política, pasando los resultados obtenidos en la medida a un *Marker* (que podrá cambiar el *DSCP* a un valor diferente) o a un *Shaper*/*Dropper* (que será el encargado de tirar o espaciar los paquetes).

2.2.3.3.2 Marcadores *(*Markers*)*

Los Marcadores de Paquetes *(Packet Markers)* ponen el campo *DS* de un paquete a un *DSCP* particular, añadiendo el paquete a un *DS behavior aggregate* concreto. El *marker* se puede configurar para marcar todos los paquetes dirigidos a él con un único *DSCP*, o bien para marcar un paquete con un valor de *DSCP* determinado dentro de un grupo de *DSCPs*, es decir, seleccionar un *PHB* determinado dentro de un grupo de *PHBs*, de acuerdo con el estado de un medidor. Cuando el *marker* cambia el *DSCP* de un paquete se dice que ha hecho un re-marcado del paquete.

2.2.3.3.3 Espaciadores *(Shapers)*

Los espaciadores del tráfico *(Shapers)* retardan algunos o todos los paquetes de un flujo de tráfico para ajustar el flujo a un *Traffic Profile* determinado. Un *Shaper* tiene normalmente un *buffer* finito y los paquetes se pueden descartar si no hay espacio suficiente en el *buffer* para almacenar los paquetes retardados.

2.2.3.3.4 Descartadores (Droppers)

Los descartadores *(Droppers)* descartan todos o alguno de los paquetes dentro de un flujo de tráfico para ajustar el flujo a un *Traffic Profile* determinado. Nótese que el *Dropper* se puede implementar como un caso especial de *Shaper* en el cual el *buffer* tiene tamaño cero.

En la figura 2.10 se muestra la arquitectura de los *DiffServ* con todos sus elementos:

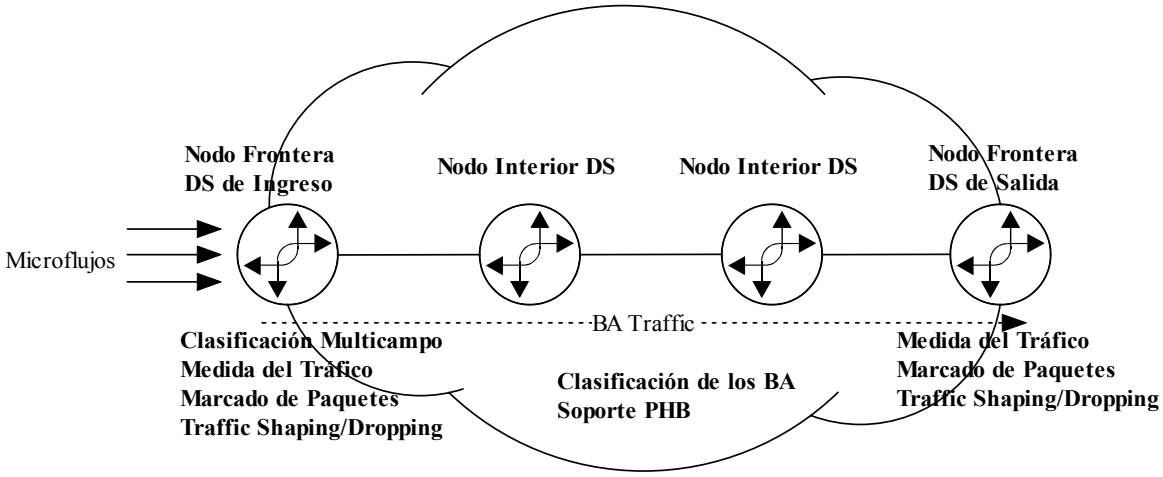

Figura 2. 10: Resumen de la Arquitectura de Servicios Diferenciados

2.2.3.4 Localización de los Traffic Conditioners y de los Multi-Field (MF) Classifiers

Los *Traffic Conditioners* y los *MF Classifiers* se encuentran habitualmente en los nodos *DS* de entrada y salida, pero también se pueden hallar en los nodos interiores de un dominio *DS*, o dentro de un dominio *non-DS-capable*.

2.2.3.4.1 Dentro del Dominio Fuente

El dominio fuente es aquel que contiene al nodo que origina el tráfico que recibe un servicio particular. Las fuentes que originan el tráfico y los nodos intermedios dentro del dominio origen pueden realizar funciones de clasificación y condicionado de tráfico. El tráfico originado desde el dominio fuente se puede marcar en las fuentes del tráfico directamente o en los nodos intermedios antes de abandonar el dominio fuente. Esto se conoce como marcado inicial o premarcado.

Hay algunas ventajas al marcar los paquetes cerca de la fuente de tráfico. Primero, una fuente de tráfico puede tener en cuenta más fácilmente las precedencias de las aplicaciones a la hora de decidir la precedencia de los paquetes. Además, la clasificación de paquetes es mucho más simple cuantos menos paquetes de fuentes diferentes se deban clasificar, reduciendo así el número de reglas en los nodos.

2.2.3.4.2 En la frontera de un Domino DS

El flujo de tráfico se puede clasificar, marcar y por otro lado, acondicionar en ambos extremos de un enlace frontera (en el nodo *DS* de entrada del domino o en el nodo *DS* de salida del domino).

El *SLA* entre los dominios debe especificar qué dominio tiene la responsabilidad de mapear los flujos de tráfico a los *DS behavior aggregates* y acondicionarlos de acuerdo con el *TCA* apropiado. Sin embargo, un nodo *DS* de entrada debe asumir que el tráfico entrante puede no estar conforme con el *TCA* y debe estar preparado para aplicar el *TCA* de acuerdo con la política local.

Si un nodo *DS* de entrada está conectado al flujo de subida de un dominio que no implementa los Servicios Diferenciados *(non-DS-capable domain)*, el nodo *DS* de entrada debe poder realizar todas las condiciones de tráfico que sean necesarias sobre el tráfico de entrada.

2.2.3.4.3 En dominios que no implementan los Servicios Diferenciados *(non-DS-capable Domanins)*

Las fuentes de tráfico o los nodos intermedios de un dominio *non-DS-capable* pueden emplear acondicionadores de tráfico para premarcar el tráfico antes de alcanzar el nodo de entrada de un dominio *DS*. En este caso las políticas locales de clasificación y marcado se pueden ocultar.

2.2.3.4.4 En los nodos interiores DS

Aunque la arquitectura básica asume que las funciones de clasificación y acondicionado del tráfico complejas se localizarán sólo en los nodos frontera de entrada y salida de una red, el despliegue de estas funciones en el interior de la red no es imposible.

2.2.4 Per-Hop Behaviors (*PHB*)

Formalmente, como viene especificado en el *RFC 2475* [5] un *per-hop behavior (PHB)* es una descripción del comportamiento de encaminado observable externamente que un nodo *DS* aplica a un *DS behavior aggregate (BA)* particular.

Como hemos visto anteriormente un *BA* es la colección de paquetes que llevan el mismo valor de *DSCP* y van en una misma dirección. Así, en términos más concretos un *PHB* se referirá al tratamiento particular que un nodo realiza sobre los paquetes correspondientes a un *BA*. Es decir, un *PHB* se refiere a la programación *(scheduling)*, encolado *(queueing)*, funciones policía *(policing)* o espaciado *(shaping)* de paquetes, que un nodo realiza sobre algunos paquetes correspondientes a un *BA* y vendrá configurado por un *SLA* o una política.

Hasta la fecha existen cuatro *PHBs* estándar [8] que son: el *PHB* por defecto *(The default PHB)*, el *PHB* selector de clase *(Class-selector PHBs)*, el *PHB* de encaminamiento asegurado *(Assured Forwarding (AFxy) PHB)* [9] y el *PHB* de encaminamiento rápido *Expedited Forwarding (EF) PHB* [10][11].

2.2.4.1 El *PHB* por Defecto

El *PHB* por defecto especifica esencialmente que un paquete marcado con un valor (recomendado) de *DSCP* de ‘000000’ obtendrá el servicio tradicional *Best Effort* de un nodo *DS-compliant*. Además, si un paquete llega a un nodo *DS-compliant* y su valor del *DSCP* no está mapeado a alguno de los otros *PHBs*, podrá ser mapeado al *PHB* por defecto.

2.2.4.2 Class-selector *PHB*s

Para preservar la compatibilidad con el anterior diseño *IP Precedence*, los valores *DSCP* son de la forma ‘xxx000’. Donde x es un valor entre 0 y 1. Estos codepoints son llamados *Class-Selector codepoints*. Obsérvese que el *PHB* por defecto es también un *Class-Selector Codepoint* (‘000000’). El *PHB* asociado con un *Class-Selector Codepoint* es un *Class-selector PHB*. Estos *PHBs* tienen casi el mismo comportamiento que los nodos que implementan *IP Precedence* basados en la clasificación y encaminamiento. Como ejemplo, paquetes con un valor de *DSCP* de ‘110000’ (*IP Precedence* 110) tienen un tratamiento preferente a la hora de encaminarlos que los paquetes con un *DSCP* de (‘100000’) (*IP Precedence* 100). Estos *PHB*s aseguran que los nodos *DS compliant* pueden coexistir con los nodos *IP-Precedence* anteriores, con la excepción de los *DTS* bits.

2.2.4.3 *Expedited Forwarding* (EF) *PHB*

Así como *RSVP*, a través del modelo de los servicios integrados, garantizaba un ancho de banda, el *EF PHB* es el ingrediente clave de los Servicios Diferenciados para ofrecer pocas pérdidas, baja latencia y baja varianza del retardo *(jitter)* y asegurar un buen ancho de banda. Aplicaciones como voz sobre *IP* (*VoIP*), video y programas de comercio electrónico requieren este tratamiento robusto por parte de la red. Aunque *EF* se puede usar dentro de los Servicios Diferenciados para ofrecer un servicio *Premium,* estará más indicado para su uso en aplicaciones críticas, con el tráfico que requiera la más alta prioridad. Está especialmente indicado para aplicaciones (como *VoIP*) que requieren muy pocas pérdidas de paquetes, ancho de banda garantizado, bajo retardo y bajo *jitter*. El valor de *DSCP* recomendado para *EF* es '101110'.

2.2.4.4 *Assured Forwarding* (AFxy) *PHB*

La tosca equivalencia con el servicio de control de carga de los servicios integrados *(IntServ Controlled Load Service)* es el *Assured Forwarding PHB*. Define un método por el cual los *BA* pueden recibir diferentes tipos de encaminamiento. Por ejemplo: el tráfico se puede dividir en clases: oro, plata y bronce. Asegurando a la clase oro un 50% del ancho de banda del enlace, a la plata un 30% y para bronce un 20%.

El *AFxy PHB* define cuatro clases *AFx*: llamadas *AF1*, *AF2*, *AF3* y *AF4*. A cada clase se le asigna cierta cantidad de *buffer* y ancho de banda de la interfaz, dependiendo del *SLA* establecido con el proveedor de servicios. Dentro de cada clase *AFx*, es posible especificar 3 valores de descarte. Así, en caso de congestión en un nodo *DS* de un enlace específico y para los paquetes que deben ser descartados de una clase *AFx* particular (pongamos *AF1*), éstos serán descartados de la siguiente manera: los marcados como *dP*(*AFx1*) (dP = drop probability, probabilidad de descarte) tendrán menos posibilidades de descartarse que los marcados con *dP(AFx2)* y éstos a su vez tendrán menos posibilidades de ser tirados que los marcados con *dP(AFx3)*. Así, la 'y' en una clase *AFxy* especifica la probabilidad de tirar un paquete dentro de una misma clase. Este concepto de probabilidad de descarte de paquetes se usa, por ejemplo, para penalizar los flujos de tráfico que dentro de un mismo *BA* excedan el ancho de banda asignado. Los paquetes de estos flujos pueden se pueden remarcar mediante un *policer* a un valor de *AF* con mayores probabilidades de ser descartado. La siguiente tabla muestra los valores *DSCP* para cada clase y probabilidad de descarte. La clase *AFx* se puede denotar por el *DSCP* 'xyzab0', donde 'xyz' es 001/010/011/100 y 'ab' representan la probabilidad de descarte.

Tabla 2. 1: Valores del campo DSCP y sus correspondientes valores de precedencia de descarte para cada clase AF PHB.

Precedencia de Descarte	Clase 1	Clase 2	Clase 3	Clase 4
Precedencia de Descarte Baja	001010	010010	011010	100010
Precedencia de Descarte Media	001100	010100	011100	100100
Precedencia de Descarte Alta	001110	010110	011110	100110

2.3 Gestión y Control activo de la Congestión

Como se ha visto, en los nodos frontera e interiores de un dominio de Servicios Diferenciados se llevan a cabo, entre otras, funciones de encolamiento para acondicionar el tráfico entrante o saliente del domino. Además, mediante estas funciones encolamiento los nodos DS pueden definir el tratamiento particular que recibirán los paquetes de un flujo de tráfico en particular. Por esto, se va a llevar a cabo un estudio de los diferentes mecanismos de colas que se pueden implementar en los nodos de una red.

La congestión en la red se produce cuando los paquetes llegan a un puerto más rápido de lo que pueden ser transmitidos. Hay dos clases de algoritmos generales que los *routers* emplean para controlar la congestión en la red [41]:

- **El servidor de cola *(queue scheduling)*** [14] gestiona la cantidad de ancho de banda reservado para cada clase de servicio en un puerto de salida. El servidor de la cola permite controlar el acceso de las clases de servicio a unos recursos limitados de red (ancho de banda del enlace). El servidor de cola decide cuándo y qué paquetes se sacan de una cola y se colocan en la interfaz de salida.

- **El gestor de la memoria de la cola (*queue memory management)*** [15] controla el número de paquetes de una cola (la profundidad de la cola), determinando cuándo y qué paquetes se descartan cuando se experimenta congestión o incluso antes de que se experimente. El gestor de la memoria permite controlar el acceso de las clases de servicio a los limitados recursos del *router* (*buffer* de memoria de los paquetes).

Mientras que estos dos mecanismos están estrechamente relacionados, tienen algunas diferencias fundamentales. El servidor de cola permite controlar la congestión controlando la reserva de la cantidad de ancho de banda del puerto de salida de las diferentes clases de servicio. El gestor de memoria intenta "evitar la congestión" *(congestion avoidance)* controlando la longitud media de las colas de paquetes.

2.3.1 Gestión de la Congestión: Disciplinas para Servir Colas

Como se ha visto, el servidor de cola decide cuándo y qué paquetes se sacan de una cola y se colocan en la interfaz de salida, es decir, es el encargado de manejar el acceso al ancho de banda de la interfaz de salida. Hay varios mecanismos para servir de colas, entre los que están:

- *First-in, First-out (FIFO) Queueing*
- *Priority Queueing (PQ)*
- *Fair Queueing (FQ)*
- *Weighted Fair Queuing (WFQ)*

2.3.1.1 *FIFO* First-in, First-out Queueing

El encolamiento de primero en entrar, primero en salir *(First-in, first-out. FIFO),* es la disciplina más básica para servir paquetes. Todos los paquetes se tratan igual, colocándolos en una única cola y sirviéndolos en el mismo orden en el que fueron colocados en ella. *FIFO* se denomina también primero en llegar, primero en servirse *(First-come,first-served, FCFS).*

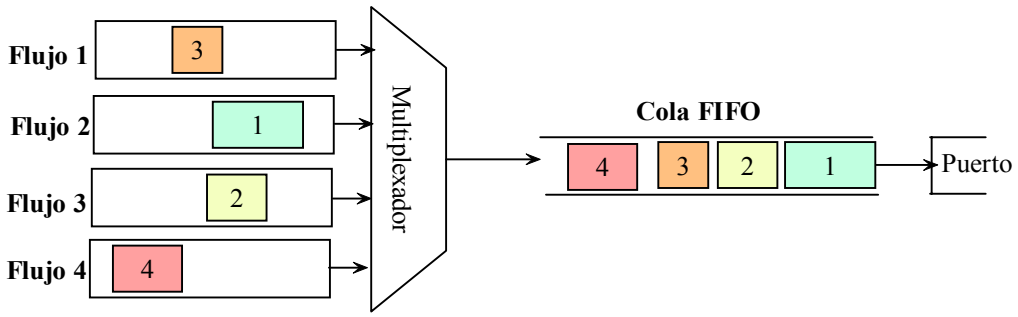

Figura 2. 11: Funcionamiento de *FIFO*

FIFO ofrece los siguientes beneficios:

- Para *routers* basados en software, el encolamiento *FIFO* supone una carga mucho menor en el sistema en comparación con disciplinas de servicio de colas más elaboradas.

- El comportamiento de una cola *FIFO* es muy predecible. Los paquetes no son reordenados y el retardo máximo viene determinado por el tamaño máximo de la cola.

FIFO también tiene las siguientes limitaciones:

- Una única cola *FIFO* no permite a los *routers* organizar los paquetes almacenados y servir una clase de tráfico de modo diferente a otras.

- Una única cola *FIFO* impacta en todos los flujos de igual modo, porque cuando se incrementa el retardo debido a la congestión, lo hace para todos los flujos igual. Como resultado, el encolamiento *FIFO* puede acabar incrementando el retardo *(delay)*, la varianza del retardo *(jitter)* y las pérdidas *(loss)* en aplicaciones de tiempo real que atraviesan una cola *FIFO*.

- Durante periodos de congestión, el encolamiento *FIFO* beneficia a los flujos *UDP* sobre los *TCP*. Cuando se pierden paquetes, debido a la congestión, las aplicaciones basadas en *TCP* reducen su tasa de transmisión pero las aplicaciones basadas en *UDP* no se enteran de la pérdida del paquete y continúan transmitiendo paquetes a su tasa usual. Debido a que las aplicaciones basadas en *TCP* ralentizan su tasa de transmisión para adaptarse a los cambios en las condiciones de la red, el encolamiento *FIFO* puede ocasionar incrementos en el retardo *(delay)*, en la varianza del retardo *(jitter)* y en una reducción de la cantidad del ancho de banda consumido por las aplicaciones *TCP* que atraviesan el *router*.

- Un flujo de ráfaga puede consumir por completo el espacio de las memorias de una cola *FIFO* y esto produce, que al resto de los flujos, se les niegue el servicio hasta que la ráfaga se sirva. Esto también provocará un incremento del retardo *(delay)*, de la varianza del retardo *(jitter)* y de las pérdidas *(loss)* de otros flujos *TCP* y *UDP* cuyo comportamiento sea correcto.

Implementaciones de *FIFO* y sus aplicaciones:

- Generalmente, el encolamiento *FIFO* se usa en un puerto de salida cuando no están configuradas otras disciplinas para servir colas. En algunos casos, los vendedores de *routers*, implementan dos colas en un puerto de salida cuando no hay configurada otra disciplina para servir cola: una cola de alta prioridad que está dedicada a servir el tráfico de control de la red y una cola *FIFO* que sirve los demás tipos de tráfico.

2.3.1.2 Priority Queueing (*PQ*)

Priority Queueing (PQ) es la base de una clase de algoritmos para servir colas, diseñados para proporcionar un método simple que soporte las clases de Servicios Diferenciados. En el *PQ* clásico, primero el sistema clasifica los paquetes y después los coloca en diferentes colas prioritarias.

Los paquetes se sirven de la cabecera de una cola, sólo, si todas las colas de prioridad mayor están vacías. Dentro de cada una de las colas de prioridad, los paquetes se sirven en el orden *FIFO*.

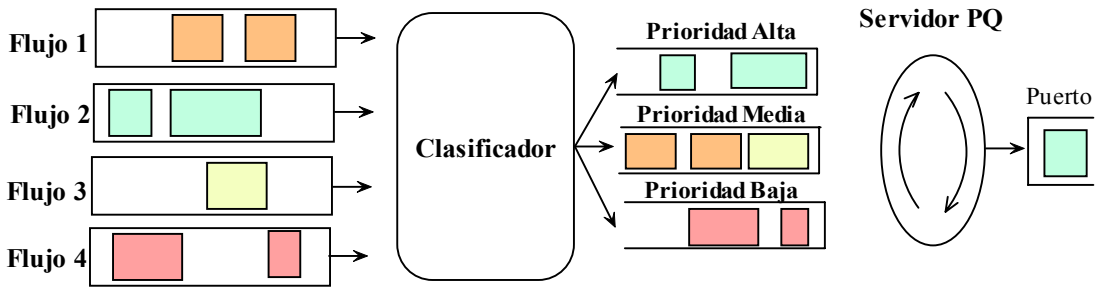

Figura 2. 12: Funcionamiento de *Priority Queueing*

PQ ofrece una par de beneficios:

- Para *routers* basados en *software*, el encolamiento *PQ* supone una carga mucho menor en el sistema en comparación con disciplinas de servicio de colas más elaboradas.

- *PQ* permite a los *routers* organizar los paquetes almacenados y por tanto servir una clase de tráfico de modo diferente a otras. Por ejemplo, se pueden colocar prioridades a las aplicaciones de tiempo real, como voz y video interactivo, y que se traten de forma prioritaria frente a otras aplicaciones que no operan en tiempo real.

Pero *PQ* también tiene varias limitaciones:

- Si la cantidad de tráfico de alta prioridad no se acondiciona o se le aplican funciones policía en los *routers* de entrada de la red, el tráfico de baja prioridad, puede experimentar un retardo excesivo mientras espera a que se sirva el tráfico de alta prioridad.

- Si el volumen de tráfico de alta prioridad llega a ser excesivo, se puede descartar el tráfico de baja prioridad cuando las memorias reservadas para este tipo de tráfico se desborden.

- Un mal comportamiento de un flujo de tráfico de alta prioridad, puede añadir un aumento significativo del retardo y de la varianza del retardo *(jitter)* experimentado por otros flujos de tráfico de alta prioridad con los que comparte la cola.

- *PQ* no es la solución a las limitaciones del encolamiento *FIFO* en donde se favorecían a los flujos *UDP* sobre los *TCP*, durante periodos de congestión.

Normalmente los vendedores de *routers* permiten configurar *PQ* para operar en uno de los dos siguientes modos: ***Strict priority queueing*** y ***Rate-controlled priority queueing***.

- ***Strict PQ*** asegura que los paquetes con una prioridad más alta en la cola, serán servidos siempre antes que paquetes en colas con más baja prioridad. Por supuesto, el problema está, como hemos visto, en que los flujos de tráfico de alta prioridad sean excesivamente elevados. Sin embargo, algunos proveedores de servicios quieren que sus redes soporten este tipo de comportamientos actualmente. Por ejemplo, un cliente puede querer usar el servicio de voz sobre *IP* y no quiere que el proveedor de servicios le tire ningún paquete para mantener cierta calidad de servicio en la comunicación. Esto se consigue usando *Strict PQ* sin limitaciones en el ancho de banda.

- ***Rate-controlled PQ*** permite a los paquetes de una cola de alta prioridad, servirse antes que paquetes de colas de menor prioridad, sólo, si la cantidad de tráfico en la cola de

mayor prioridad permanece por debajo de un umbral configurado por el usuario. Por ejemplo, se asume que la cola de mayor prioridad tiene una tasa límite del 20% del ancho de banda del puerto de salida. Mientras que la cola de mayor prioridad consuma menos del 20% del ancho de banda del puerto de salida los paquetes de esta cola se servirán antes que los paquetes de colas con una prioridad menor. Sin embargo, si sobrepasa ese límite, los paquetes de menor prioridad pueden servirse antes.

Hay dos aplicaciones principales para *PQ* en la entrada y en el interior de una red:

- *PQ* puede aumentar la estabilidad de la red durante periodos de congestión, permitiendo asignar protocolos de *routing* y otros tipos de tráfico de control de la red a las colas con la prioridad mayor.

- *PQ* soporta la entrega de clases de servicio de alto caudal, bajo retardo *(low-delay)*, varianza del retardo pequeña *(low-jitter)* y pocas pérdidas *(low-loss)*. Esta capacidad permite proporcionar un mejor servicio a aplicaciones de tiempo real.

Sin embargo, para que se soporten estos tipos de servicio, será necesario que se acondicione el tráfico en las entradas de la red.

2.3.1.3 Fair Queueing (FQ)

FQ fue propuesto por John Nagle en 1987. *FQ* está diseñado para asegurar que cada flujo tenga un acceso justo a los recursos de la red y evita que un flujo de ráfagas consuma más ancho de banda que la parte que le corresponde. En *FQ*, primero el sistema clasifica los paquetes en flujos y los asigna a una cola dedicada especialmente para ese flujo. Las colas se sirven siguiendo un tiempo en orden *round-robin*, es decir, en orden secuencial circular (del primero al último y vuelta al primero). Las colas vacías se saltan. *FQ* se denomina también *per-flow* o *flow-based queueing*.

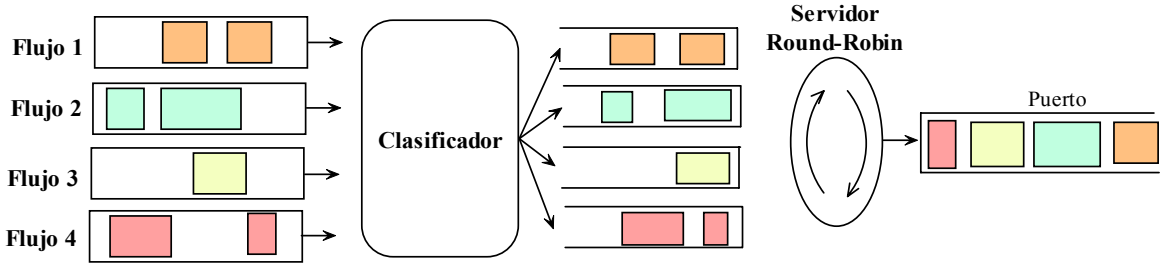

Figura 2. 13: Funcionamiento de *Fair Queueing*

Beneficios de *FQ*:

- El primer beneficio de *FQ* es que un flujo con demasiadas ráfagas o un flujo que no colabore no degradarán la calidad de servicio que reciban otros flujos debido a que se aísla a cada flujo en su propia cola.

- Si un flujo intenta consumir más de su ancho de banda, esto sólo afectará a su cola y por lo tanto no influirá en la ejecución de las otras colas.

FQ también tiene algunas limitaciones:

- Las implementaciones de *FQ* de los proveedores están implementadas en software no en hardware. Esto limita la aplicación de *FQ* a interfaces de baja velocidad en las entradas de la red.

- El objetivo de *FQ* es reservar la misma cantidad de ancho de banda a cada flujo. *FQ* no está diseñado para soportar un número de flujos con diferentes requerimientos de ancho de banda.

- *FQ* proporciona cantidades iguales de ancho de banda a cada flujo sólo si todos los paquetes de todas las colas tienen el mismo tamaño. Los flujos que contienen paquetes de mayor tamaño obtienen una porción del ancho de banda de salida mayor.

- *FQ* es sensible al orden de llegada de los paquetes. Si un paquete llega a una cola vacía inmediatamente después de que la cola sea visitada por el servidor *round-robin*, el paquete tendrá que esperar en la cola hasta que todas las otras colas se sirvan antes de poder ser transmitido.

- *FQ* no proporciona un mecanismo que permita implementar fácilmente servicios de tiempo real como *VoIP*.

- *FQ* asume que se puede clasificar el tráfico de la red en flujos bien definidos fácilmente. En una red *IP* esto no es tan fácil como parece. Se pueden clasificar flujos basándose en la dirección de origen de un paquete pero entonces a cada estación de trabajo se le proporcionan los mismos recursos de red que a una estación servidor.

- *FQ* depende del mecanismo específico que utilices para clasificar los paquetes en flujos, generalmente *FQ* no se puede configurar en *routers* interiores debido a que los *routers* interiores requerirían miles o cientos de miles de colas discretas en cada puerto.

Implementaciones de *FQ* y sus aplicaciones:

- *FQ* se aplica normalmente en las entradas de una red, donde los clientes se conectan con sus proveedores de servicio. Las implementaciones de *FQ* de los vendedores normalmente clasifican los paquetes en 256, 512 o 1024 colas empleando una función de *hash* que se calcula con las parejas de direcciones de origen y destino, los puertos de *TCP* origen y destino y el byte *IP* de tipo de servicio *ToS*. *FQ* requiere una configuración mínima (sólo activarlo o desactivarlo). Cuando el número de colas cambia el ancho de banda reservado para cada cola también cambia. Por ejemplo, si el número de colas se incrementa de n a n+1 entonces la cantidad de ancho de banda reservado para cada cola se decrementa de 1/n a 1/n+1.

- *FQ* proporciona un aislamiento ideal para flujos de tráfico individuales ya que en las entradas de la red un cliente normal tiene un número de flujos limitado. Esto reduce el impacto que el mal comportamiento que un flujo pueda tener sobre los otros flujos que atraviesan el mismo puerto de salida.

- En ***FQ* basado en clase *(Class-Based FQ)*** el puerto de salida se divide en un número de diferentes clases de servicio. Para cada clase de servicio se reserva un porcentaje del ancho de banda de salida configurado por el usuario. Entonces se aplica *FQ* dentro del bloque de ancho de banda reservado para cada clase de servicio. Como resultado, todos los flujos asignados a una clase de servicio dada proporcionan un reparto igual del conjunto de ancho de banda configurado para esa clase de tráfico en particular.

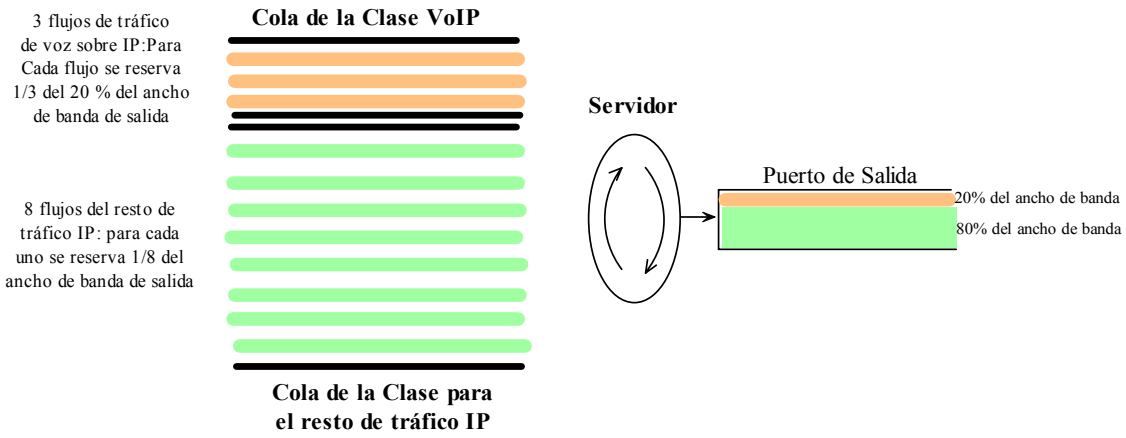

Figura 2. 14: Funcionamiento de Class-Based *FQ*

En la figura anterior asumimos que hay dos clases de servicio configuradas para un puerto de salida. Para *VoIP* se reserva el 20% del ancho de banda del enlace de salida y para el resto del tráfico *IP* el 80%. En el modelo *FQ* basado en clase para cada flujo de *VoIP* se reserva 1/3 del 20% anterior (bloque de ancho de banda) y para cada flujo del resto de tráfico *IP* 1/8 del 80% del ancho de banda de salida.

2.3.1.4 Weighted Fair Queuing (*WFQ*)

WFQ [14][24][29][30] fue creado en 1989 por Lixia Zhang y por Alan Demers, Srinivasan Keshav y Scott Shenke independientemente. *WFQ* es la base de un tipo de disciplinas para servir colas diseñadas para solucionar las limitaciones del modelo *FQ*:

- *WFQ* soporta flujos con diferentes requerimientos de ancho de banda. Esto lo logra dándole a cada cola un peso que le asigna un porcentaje diferente del ancho de banda de salida.

- *WFQ* también soporta paquetes de longitud variable de forma que los flujos con paquetes mayores no dispongan de un ancho de banda mayor que los flujos cuyos paquetes sean de menor tamaño. Esto añade una mayor complejidad a los algoritmos de servicio de colas. Por ello, estas disciplinas de servicio de colas funcionan mejor con paquetes de longitud fija (redes *ATM* basadas en celdas) que con paquetes de longitud variable (redes *IP*).

WFQ soporta la distribución equitativa del ancho de banda de paquetes de longitud variable mediante la aproximación a un sistema de procesador compartido generalizado *Generalized Processor Sharing* (*GPS*). Mientras que *GPS* es un servidor teórico que no se puede implementar, su comportamiento es similar a la disciplina de servicio basada en *weighted bit-by-bit round-robin*. Esta aproximación soporta la reserva equitativa del ancho de banda debido a que tiene en cuenta la longitud del paquete. Como resultado, en algún momento, cada cola recibe su porción del ancho de banda configurado.

La siguiente figura muestra una disciplina de servicio basada en weighted bit-by-bit round-robin sirviendo tres colas. Se asume que a la cola 1 se le asigna el 50% del ancho de banda del enlace de salida y que a la cola 2 y 3 se les asignan el 25% del ancho de banda. El mecanismo de servicio transmite dos bits de la cola 1, uno de la cola 2 y uno de la cola 3 y vuelve de nuevo a la cola 1. Como resultado de la disciplina basada en peso el último bit del paquete de 600 bytes se transmite antes del último bit del paquete de 350 bytes y el último bit del paquete de 350

bytes se transmite antes que el último bit del paquete de 450 bytes. Esto produce que el paquete de 600 bytes acabe (sea reensamblado completamente) antes del paquete de 350 bytes y el paquete de 350 bytes termine antes que el paquete de 450 bytes.

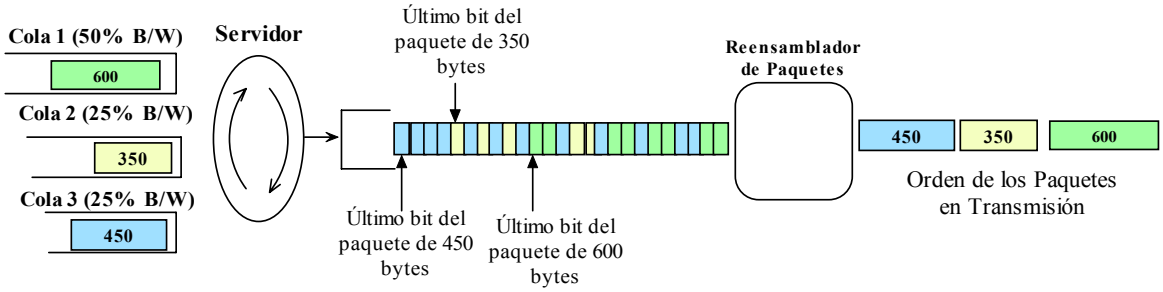

Figura 2. 15: Disciplina de Servicio Basada en Weighted Bit-by-Bit Round-Robin Sirviendo tres Colas.

WFQ se aproxima a este servicio teórico de colas calculando y asignando un tiempo de fin a cada paquete. Dadas la tasa de bit del puerto de salida, el número de colas activas, el peso relativo asignado a cada cola y la longitud de cada paquete en cada cola, la disciplina de servicio calcula y asigna un tiempo de fin a cada paquete que llega. El servidor o *scheduler* entonces selecciona y encamina el paquete que acaba antes de todos. Es muy importante entender que el tiempo de fin, no es el tiempo actual de transmisión de cada paquete. Así, el tiempo de fin del paquete es un número asignado a cada paquete que representa el orden en el que el paquete se transmitirá por el puerto de salida.

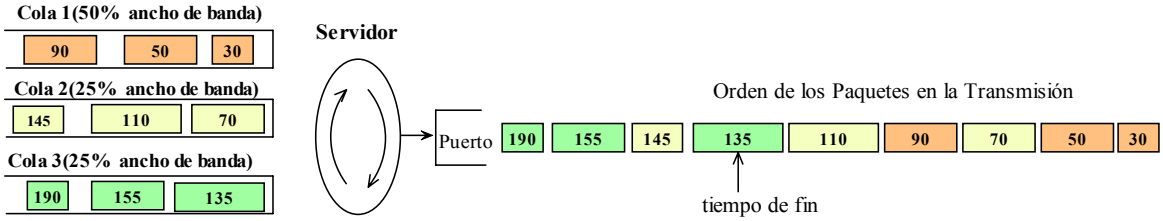

Figura 2. 16: *WFQ* Calculando y Asignando un Tiempo de Fin a Cada Paquete.

Cuando cada paquete se clasifica y se coloca en la cola, el servidor de la cola calcula y asigna un tiempo de fin a cada paquete. Cuando el servidor *WFQ* sirve sus colas, selecciona el paquete con el tiempo de fin menor como el próximo paquete a transmitir por el puerto de salida. Por ejemplo, si *WFQ* determina que el paquete A tiene un tiempo de fin de 30, el paquete B tiene un tiempo de fin de 70 y el paquete C tiene un tiempo de fin de 135, entonces el paquete A se transmitirá antes que el paquete B o que el paquete C. Observar que con los pesos adecuados *WFQ* puede transmitir dos o más paquetes consecutivos de una misma cola.

WFQ tiene dos beneficios principalmente:

- Proporciona la protección de cada clase de servicio asegurando un nivel mínimo del ancho de banda del puerto de salida independientemente del comportamiento de otras clases de servicio.

- Cuando se combina con acondicionadores de tráfico en las entradas de una red, *WFQ* garantiza un reparto equitativo del ancho de banda del puerto de salida de cada clase de servicio con un retardo limitado.

Sin embargo *WFQ* también tiene varias limitaciones:

- Las implementaciones de los proveedores de *WFQ* están realizadas en software no en hardware. Esto limita la aplicación de *WFQ* a interfaces de velocidad baja en las entradas de la red.

- *WFQ* implementa un algoritmo complejo que requiere el mantenimiento de una cantidad significativa de estados de clases de servicio y escaneos interactivos del estado en cada paquete que llega.

- La complejidad computacional impacta en la escalabilidad de *WFQ* cuando intenta mantener un gran número de clases de servicio en interfaces de alta velocidad.

- En interfaces de alta velocidad puede que no compense minimizar el retardo de un único paquete con el alto gasto computacional, si se considera la insignificante cantidad de retardo de serialización introducido por los enlaces de alta velocidad y los pocos requerimientos computacionales de otras disciplinas de servicio de colas.

Desde que fue propuesto inicialmente en 1989, se han realizado muchas modificaciones de *WFQ* con diferentes compensaciones intentando equilibrar complejidad, exactitud y ejecución. Como resultado de esas variaciones tenemos los 4 *WFQ* siguientes:

- ***Class-Based WFQ*** asigna paquetes a colas basándose en criterios de clasificación de paquetes definidos por el usuario. Por ejemplo, los paquetes se pueden asignar a una cola en particular basándose en los bits del *IP Precedence*. Después de que los paquetes sean asignados a sus colas, podrán recibir un trato prioritario en función de los pesos configurados por el usuario.

- ***Self-clocking Fair Queueing (SCFQ)*** es una mejora a *WFQ* que simplifica la complejidad de calcular el tiempo de fin en el correspondiente sistema *GPS*.

- ***Worst-case Fair Weighted Fair Queueing (WF^{2}Q)*** es una mejora a *WFQ* que emplea los tiempos de comienzo y fin de los paquetes para alcanzar más exactitud en la simulación de un sistema *GPS*.

- ***Worst-case Fair Weighted Fair Queueing + (WF^{2}Q+)*** es una mejora a (*WF^{2}Q*) que implementa una nueva función virtual que da como resultado una disminución de la complejidad y una mayor exactitud.

Implementaciones de *WFQ* y sus aplicaciones.

- *WFQ* se puede configurar para clasificar paquetes en un número de colas relativo, empleando una función de *hash* que se calcula con las parejas de direcciones de origen y destino, los puertos de *TCP* origen y destino y el byte *IP* de tipo de servicio *ToS*.

- Se puede configurar *WFQ* para permitir al sistema servir un número limitado de colas que llevan un conjunto de flujos de tráfico. Para esta configuración el sistema utiliza políticas de *QoS* o los tres bits de *IP Precedence* de menor peso en el byte de tipo de servicio *ToS* para asignar los paquetes a las colas. Para cada una de las colas se reserva un porcentaje diferente del ancho de banda del puerto de salida basándose en el peso que el sistema calcula para cada clase de servicio. Esta aproximación le permite al sistema reservar diferentes cantidades de ancho de banda a cada cola basándose en la política de *QoS* o reservar cantidades incrementales de ancho de banda para cada cola cuando se incrementa el valor de *IP Precedence*.

- Una versión mejorada de *WFQ*, a veces denominada *WFQ* basado en clase ***(Class-Based WFQ)***, se puede emplear alternativamente para servir un número limitado de colas que lleve un conjunto de flujos de tráfico. Para esta configuración las reglas de clasificación de paquetes definidas por el usuario asignan los paquetes a colas que tienen reservado un porcentaje del ancho de banda configurado por usuario. Esta aproximación permite determinar de manera precisa que paquetes se agrupan en una clase de servicio dada y especificar la cantidad exacta de ancho de banda reservado para cada clase de servicio.

2.3.2 Evitar la Congestión: Gestión Activa de la Memoria de las Colas

Antes de comenzar la discusión de los gestores de la memoria de cola es importante revisar las principales razones de tener *buffers* de paquetes en redes multiplexadas estadísticamente. Los *buffers* de paquetes absorben las ráfagas periódicas de paquetes porque no las descartan y las pueden transmitir en periodos de inactividad. Si los *buffers* de los *routers* tienen una media de profundidad de cola estable del 20% de capacidad, tienen menos posibilidades de absorber ráfagas que si tuvieran una media de profundidad de cola cercana al 0% de la capacidad. En una red donde los paquetes llegan en ráfagas es absolutamente critico que la media de profundidad de cola este lo más cercana posible al 0% de la capacidad para absorber ráfagas sin descartar paquetes.

2.3.2.1 Tail Drop

Tail Drop significa la ausencia completa de un gestor de la memoria de la cola. Cuando un paquete llega al final de una cola completamente llena. El paquete se descarta al igual que todos los que lleguen tras él hasta que hay espacio disponible en la cola.

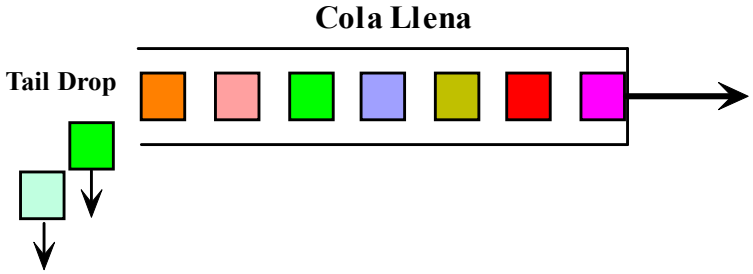

Figura 2. 17: Funcionamiento de *Tail Drop*

Los beneficios del gestor de cola *Tail Drop* incluyen:

- *Tail Drop* es fácil de implementar por los proveedores y de entender por parte de los clientes.

- *Tail Drop* puede reducir el número de los paquetes descartados con un incremento del tamaño de las colas, sin embargo esto hará aumentar el retardo de extremo a extremo de todos los flujos que atraviesen la cola.

Las limitaciones de *Tail Drop* son:

- *Tail Drop* no descarta paquetes hasta que la cola esta completamente llena y los recursos consumidos completamente. Esto significa que la cola no puede absorber

ráfagas de tráfico hasta que no haya espacio disponible en la cola. Esto puede dar como resultado un comportamiento de cierre, debido a la falta de espacio de *buffer* para almacenar los paquetes entrantes. En consecuencia un pequeño número de flujos puede monopolizar toda la capacidad del *buffer* e impedir a las sesiones existentes o nuevas acceder a la cola.

- *Tail Drop* permite a las colas permanecer llenas o casi llenas durante largos periodos de tiempo, ya que los *host* no reconocen la congestión (mediante el descarte de paquetes) hasta que las colas no alcanzan el 100% de su capacidad y se consumen completamente los recursos.

- *Tail Drop* es un algoritmo extremadamente pobre para tráfico basado en *TCP*. Aproximadamente del 85 al 95% del tráfico de las redes *IP* es *TCP*. *TCP* supone que si se tira un paquete en un *router* es debido a la congestión. Esto permite a las sesiones *TCP* controlar su propia tasa de transferencia. Sin embargo, *Tail Drop* produce que todas las sesiones *TCP* que atraviesan la cola congestionada reduzcan sus tasas de transmisión al mismo tiempo resultando un proceso conocido como sincronización global *TCP*. Esto produce oscilaciones drásticas en el tráfico que dan como resultado un uso ineficiente del ancho de banda de salida debido a que muchas sesiones dividen por dos sus ventanas de transmisión al mismo tiempo.

- Las sesiones individuales de *TCP* se recuperan más lentamente de descartes de paquetes múltiples que se un descarte individual. Esto puede reducir significativamente el caudal global de los flujos de los clientes.

- Una media de la profundidad de la cola grande incrementa el retardo de extremo a extremo experimentado por los flujos de los clientes que atraviesan la red.

2.3.2.2 Gestión Activa de la Memoria de las Colas

Tail Drop es la forma más simple de gestionar la memoria de la cola ya que representa la ausencia total de un gestor de la memoria de la cola. Los gestores de la memoria de cola activos permiten a un *router* responder a la congestión de forma activa si la media de los tamaños de sus colas comienza a incrementarse. En vez de esperar a que se congestione la cola y se desborde y realizar *Tail Drop* con todos los paquetes que lleguen, los gestores de memoria de cola activos responden a la congestión marcando o descartando los paquetes antes de que los recursos de memoria de la cola se consuman completamente.

Hay dos mecanismos que soportan la gestión activa de las memorias de las colas en grandes redes *IP*:

- Random Early Detection (*RED*) [15][42] – desplegada actualmente en la mayoría de las redes *IP*

- *Explicit Congestion Notification (ECN)* – experimental

Los beneficios de la gestión activa de las colas comparadas con *Tail Drop* incluyen:

- La eliminación de la sincronización global de fuentes *TCP* que da como resultado un uso más eficiente del ancho de banda de la red.

- El soporte de fluctuaciones momentáneas en el tamaño de la cola, que permiten absorber ráfagas sin descartar paquetes y causar que los *host* reduzcan sus caudales cuando reducen sus tasas de transmisión.

- La habilidad para controlar el tamaño de la cola influyendo en la media del retardo de encolamiento a través del *router*.

2.3.2.2.1 Random Early Detection (*RED*)

A diferencia de *Tail Drop* que no proporciona una gestión de la cola, *RED* es un gestor de cola activo desplegado actualmente en numerosas redes *IP*. Con *RED* el descarte de un único paquete es suficiente señal de congestión para los *host* que usan *TCP*. Al descartar un sólo paquete un *router* envía una advertencia implícita a una fuente *TCP* de que el paquete descartado ha experimentado congestión en algún punto a lo largo del camino de extremo a extremo. Como respuesta a esta advertencia implícita, la fuente *TCP* reduce su tasa de transmisión para que el *buffer* del *router* no se desborde.

RED emplea un perfil de descarte *(drop profile)* del paquete para controlar la agresividad del proceso de descarte de paquetes. El perfil de descarte define un rango de probabilidades de descarte mediante un rango de estados de ocupación de la cola. Si el estado de ocupación permanece por debajo de un umbral mínimo configurado por el usuario min_{th}, un paquete nunca se descartará de la cola. Si el nivel de ocupación excede un umbral máximo max_{th}, la cola funcionará como si estuviera configurado *Tail Drop*. Si el estado de ocupación de la cola permanece entre el min_{th} y el max_{th}, un paquete se tirará de acuerdo con una probabilidad definida por el usuario. Generalmente se configuran los parámetros de *RED* para mantener la ocupación media de la cola entre el min_{th} y el max_{th}

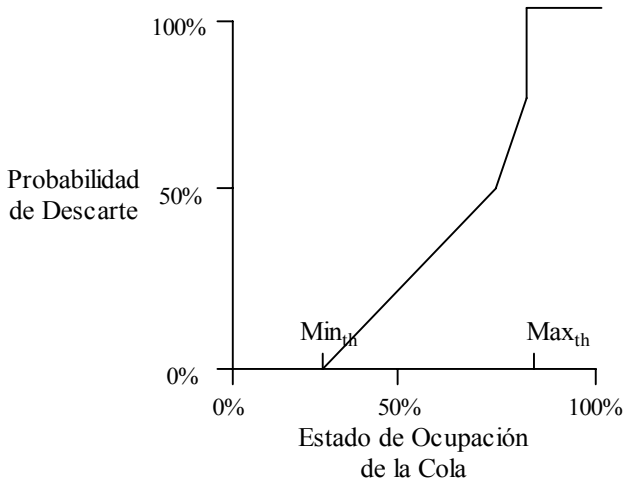

Figura 2. 18: Un Perfil de Descarte de *RED*

En la figura 2.18, si el uso de la cola es del 25% de su capacidad hay un 0% de probabilidad de que el paquete se descarte, una cola con un uso del 50% tendrá una probabilidad de 0.25 de que se descarten los paquetes, una cola con una utilización del 75% indica que hay una probabilidad de 0.5 de que se descarten los paquetes y cuando la cola está empleada más del 85% de su capacidad todos los paquetes se descartarán.

Uno de los retos para proporcionar una implementación de *RED* satisfactoria, será seleccionar el mecanismo empleado para calcular la congestión inminente. Por lo tanto, las implementaciones de *RED* se diferencian en cómo calculan el grado de ocupación de la cola. Algunas implementaciones proporcionan medidas instantáneas de la profundidad de la cola. Otras implementaciones utilizan diferentes algoritmos de peso-medio para determinar la profundidad de la cola en periodos de tiempo.

Los principales beneficios de *RED*:

- *RED* no requiere cambios en los protocolos de *TCP* actuales.

- *RED* identifica las etapas tempranas de congestión y responde con descartes aleatorios de paquetes. Si la cantidad de congestión se sigue incrementando, *RED* descarta paquetes de manera más agresiva para evitar que la cola alcance el 100% de su capacidad, que daría como resultado una denegación completa del servicio. Esto permite a *RED* mantener un límite superior en la profundidad media de la cola incluso con protocolos de la capa de transporte no cooperantes.

- Debido a que *RED* no espera hasta que la cola se llene para comenzar a descartar paquetes, *RED* permite a la cola aceptar ráfagas de tráfico y no descartar todos los paquetes de una ráfaga. Como resultado, *RED* trata bien al tráfico *TCP* ya que no descarta muchos paquetes de una misma sesión *TCP* y ayuda a evitar la sincronización global de *TCP*.

- *RED* permite mantener la cantidad de tráfico en una cola en un nivel moderado. Ahora, el ancho de banda de salida no estará infrautilizado. *RED* permite mantener la profundidad de la cola en un nivel que produce la mejor utilización del ancho de banda de salida.

- *RED* soporta el descarte equitativo de paquetes de múltiples flujos sin necesidad de que el *router* mantenga estado de la cantidad de tráfico que tiene cada flujo que atraviesa una cola dada. Por ejemplo, si tienes configurado *RED* con una probabilidad de descarte de 0.2 cuando la cola esté al 50% de su capacidad, significa que uno de cada 5 paquetes se descartará cuando la cola alcance el 50% de su capacidad. Este perfil de descarte afectará más a un flujo cuyos paquetes suponen el 40% de todos los paquetes de la cola que a un flujo que sus paquetes signifiquen el 5%.

Las limitaciones de *RED*:

- *RED* puede ser difícil de configurar si se quiere alcanzar una ejecución predecible.

- Hay una falta de experiencia operacional con los parámetros de configuración específicos de *RED* que trabajan mejor que *Tail Drop*.

- Si no se ponen los parámetros de configuración adecuados de *RED* puede que la utilización del ancho de banda de salida sea peor que si se usa *Tail Drop*.

- *RED* influye sobre los flujos *TCP* pero no sobre los flujos que no son *TCP* (como mensajes del protocolo de control de Internet (*ICMP*) y *UDP*). Cuando se tira un paquete que no es de *TCP* con *RED* la fuente no sabe que el paquete se ha tirado y no altera su tasa de transmisión. Por esta razón se recomienda no usar *RED* con tráfico basado en *UDP*. También se recomienda utilizar tamaños de cola pequeños para este tipo de tráfico para evitar grandes retardos.

2.3.2.2.2 Weighted Random Early Detection (*WRED*)

WRED [15][25][30][31] es una extensión de *RED* que permite asignar diferentes perfiles de descarte *RED* a diferentes tipos de tráfico. La habilidad para definir diferentes perfiles de descarte a diferentes colas o a diferentes tipos de tráfico en la misma cola proporciona una precisión mayor de control que el *RED* clásico. Por ejemplo, suponiendo que la gestión de la memoria de la cola permitiese definir dos niveles de precedencia de descarte dentro de una misma cola. Esto permitiría asignar un perfil de descarte de *RED* menos agresivo para ciertos paquetes y más agresivo para otros dado un mismo nivel de congestión.

Además, estadísticamente *WRED* tira más paquetes de grandes que de pequeños usuarios, entendiendo por usuario de gran tamaño aquel que consume más ancho de banda o recursos de la red. Por lo tanto, las fuentes de tráfico que generan la mayoría del tráfico tienen más posibilidades de ser reducidas que las fuentes que generan menos.

2.1.1.1.1.1 Funcionamiento de *Weighted Random Early Detection WRED*

Cuando un paquete llega, ocurre lo siguiente:

- Se calcula el tamaño medio de la cola (ver tamaño medio de la cola).
- Si la media es menor que el umbral mínimo del tamaño de la cola, el paquete es encolado.
- Si la media está entre el umbral mínimo y máximo del tamaño de la cola, el paquete puede ser descartado o encolado, dependiendo de la probabilidad de tirado del paquete (ver más adelante).
- Si el tamaño medio de la cola es mayor que el umbral máximo de la cola, el paquete es automáticamente descartado.

Tamaño medio de cola

El tamaño medio de la cola se calcula en base al tamaño medio anterior y el tamaño actual de la cola. La fórmula es:

$$\text{Media} = (\text{Media\_anterior} * (1 - 1/2\text{^ n})) + (\text{tamaño\_actual} * 1/ 2\text{^ n})$$

Donde 'n' es el factor de peso exponencial *(exponential weight factor)*, configurable por el usuario.

Para valores altos de 'n', se tiene más en cuenta el tamaño medio de cola anterior. Un factor grande suaviza las crestas y bajadas en la longitud de la cola. Es improbable que el tamaño medio de la cola cambie muy rápido, evitando oscilaciones drásticas en tamaño. El proceso *WRED*, es lento en comenzar a tirar paquetes, pero puede continuar tirando paquetes después de que el tamaño actual de la cola se coloque por debajo del umbral mínimo.

Si el valor de 'n' alcanza valores demasiado altos, *WRED* no reacciona a la congestión. Los paquetes son transmitidos o tirados como si *WRED* no existiera. Mientras, para valores pequeños de 'n', el tamaño medio de la cola se parece más al tamaño actual de la cola. La media resultante puede fluctuar con los cambios en los niveles de tráfico. En este caso, el proceso *WRED* responde rápidamente cuando la cola se llena. Una vez que la cola se coloca por debajo del umbral mínimo, el proceso deja de tirar paquetes. Finalmente,si el valor de 'n' llega a ser demasiado bajo, *WRED* reacciona muy fuerte a ráfagas temporales de tráfico y tira paquetes innecesariamente.

Probabilidad de Tirado de Paquetes

La probabilidad de que un paquete sea tirado se basa en el umbral mínimo, máximo y en el *mark probability denominator*.

Cuando el tamaño medio de la cola se encuentra sobre el umbral mínimo, el algoritmo *RED* comienza a tirar paquetes. La tasa de descarte de paquetes se incrementa linealmente al igual

que el tamaño medio de la cola, hasta que el tamaño medio alcanza el umbral máximo. Cuando el tamaño medio de la cola está sobre el umbral máximo, se tiran todos los paquetes.

El valor del umbral mínimo debe ser suficientemente alto para maximizar el uso del enlace. Si el umbral mínimo es demasiado bajo, se tirarán paquetes innecesariamente, y el enlace de transmisión puede infrautilizarse.

Además, la diferencia entre el umbral máximo y el mínimo debe ser lo suficientemente grande para evitar sincronización global de los *host TCP* (puede ocurrir cuando muchos *host TCP* reducen sus tasas de transferencia). Si la diferencia entre el umbral mínimo y máximo es demasiado pequeña, se pueden tirar muchos paquetes a la vez, dando como resultado una sincronización global.

Hay varios modos en los que un proveedor puede implementar *WRED*. Los *Traffic Policing* pueden marcar paquetes que están fuera del perfil (exceden la tasa contratada) con una indicación de que tienen una probabilidad alta de ser descartados y los paquetes dentro del perfil marcarlos con una indicación de que su probabilidad de ser descartados es menor. El perfil correcto de descarte de *RED* se le puede aplicar al paquete si el *router* experimenta congestión basándose en la marca que lleva el paquete. El *router* puede marcar el tráfico *TCP* de modo que el sistema de gestión de la cola pueda diferenciar entre paquetes *TCP* o *UDP*. Esto permite asignar un perfil de descarte de *RED* a paquetes *TCP* y uno diferente a paquetes *UDP*.

El *router* permite diferenciar gran cantidad de paquetes *TCP* dentro del perfil (*in-profile*) y fuera del perfil *(out-of-profile)* y paquetes *UDP* dentro del perfil (*in-profile*) y fuera del perfil *(out-of-profile)* y aplicarles diferentes perfiles de descarte *RED* a cada uno.

Ejemplo 1: Asignando un perfil de descarte *RED* diferente a cada cola:

La habilidad de definir diferentes perfiles de descarte *RED* para cada cola en el mismo puerto de salida, permite asignar perfiles de descarte *RED* menos agresivos a colas con una prioridad alta que a colas con una prioridad menor. En la Figura 2. 19 se le asigna un perfil de descarte *RED* diferente a cada cola. Para la cola de mayor prioridad, la pendiente de la gráfica es poco pronunciada, así *RED* será menos agresivo a la hora de tirar paquetes de esta cola. Sin embargo, para la cola de más baja prioridad, la pendiente es muy pronunciada luego *RED* descartará muchos paquetes cuando sobrepase el umbral mínimo.

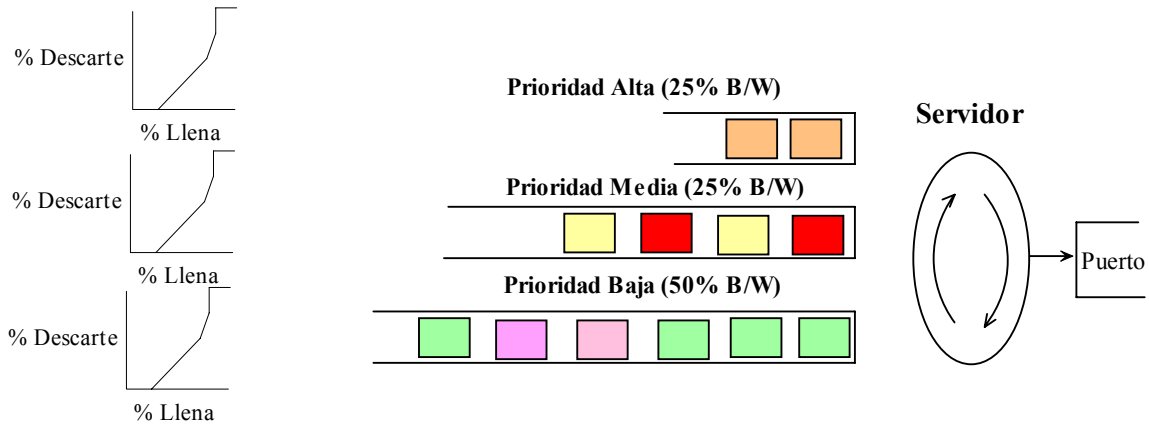

Figura 2. 19: Un solo Perfil de Descarte para cada Cola

Ejemplo 2: Asignando un conjunto diferente de perfiles de descarte *RED* a cada cola:

La habilidad de asignar un conjunto diferente de perfiles de descarte *RED* a cada cola permite aplicar un perfil de descarte *RED* para paquetes *in-profile* y otro diferente para paquetes *out-of-profile*. En la Figura 2. 20, a cada cola se le asignan dos perfiles de descarte *RED* diferentes. Así, *RED* podrá tratar de manera distinta a los paquetes de una misma clase que estén dentro y fuera del perfil de la clase. Por ejemplo, un *policer* podría encargarse de medir si los paquetes de una clase cumplen o no el contrato. Si lo cumplen marcarlos como "dentro de perfil" y si no como "fuera de perfil". Luego RED se encargaría de tratarlos de manera diferente.

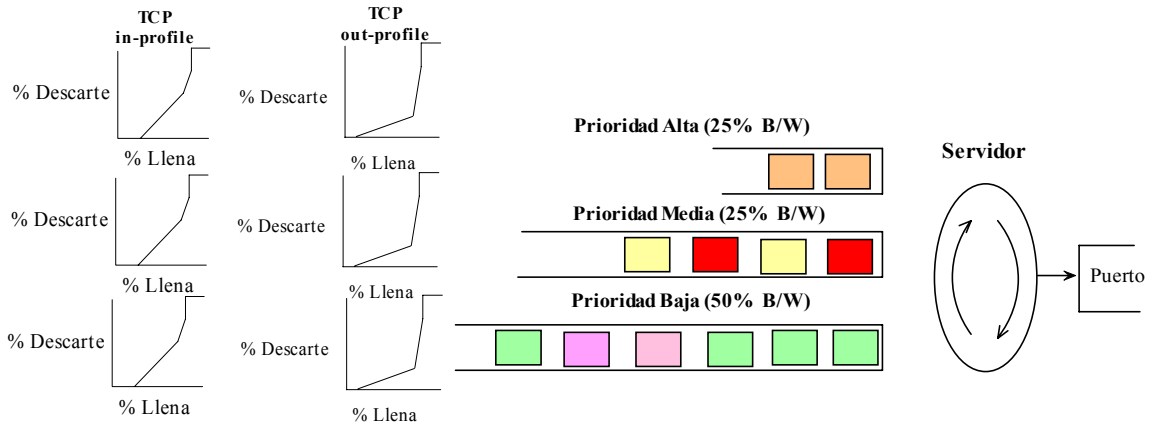

Figura 2. 20: Un Conjunto de Perfiles de Descarte para cada Cola

2.4 Traffic *Policing*

Como se ha expuesto anteriormente, en los extremos del dominio de Servicios Diferenciados actúan los acondicionadores del tráfico para adecuar el tráfico entrante a las características del dominio. Esta adecuación puede incluir reducciones en las tasas de transferencia, marcado o remarcado de los paquetes con unos determinados valores de *DSCP*, etc… Una de las funciones llevada a cabo por estos acondicionadores del tráfico es la función de policía o *Traffic Policing* [16].

Traffic Policing permite controlar la tasa máxima transmitida o recibida sobre la interfaz de un *router*. De este modo se podrá controlar el ancho de banda del enlace. *Traffic Policing* se configura frecuentemente sobre interfaces en los extremos de la red para limitar el tráfico que entra o sale de ella. En la mayoría de las configuraciones de *Traffic Policing*, el tráfico que cae dentro de los parámetros acordados es transmitido, mientras que el que excede es descartado o transmitido con una prioridad diferente. Uno de los algoritmos más utilizados en la actualidad para implementar *Traffic Policing* el denominado *Token Buket* que se explicará en la siguiente sección.

2.4.1 ¿Qué es un *Token Bucket*?

Un *Token Bucket* [39] es la definición formal de una tasa de transferencia. Tiene tres componentes: el tamaño de ráfaga (*burst size*), la tasa media (*mean rate*) y el intervalo de tiempo (*time interval*) (*T*). Aunque la tasa media se representa generalmente en bits por segundo, alguno de los dos valores puede derivar del tercero debido a la relación que se muestra a continuación:

$$\text{Tasa media} = (\text{tamaño de ráfaga})/(\text{intervalo de tiempo})$$

Definiciones de los términos:

- **Tasa media *(mean rate)*:** También llamada *Committed Information Rate (CIR)*, especifica cuantos datos pueden ser enviados o encaminados por unidad de tiempo de media.

- **Tamaño de ráfaga *(burst size)*:** También llamada *Committed Burst (Bc) size*, especifica en bits por ráfaga el tamaño del cubo.

- **Intervalo de tiempo *(time interval):*** También llamado el intervalo de medida, especifica el quantum de tiempo en segundos por ráfaga (T).

En la Figura 2. 21 se pueden ver los términos anteriores. Se puede apreciar la relación entre la tasa de llegada de los paquetes y el *CIR*. Llegan tres paquetes a un *Token Bucket*. Los dos primeros estarán conformes con el CIR y el tercero no, ya que el tercero sobrepasará el límite de datos permitidos en el tiempo T (ese límite de datos es Bc). Este tercer paquete puede ser marcado, encolado o descartado dependiendo de la política que utilice el *Token Bucket*.

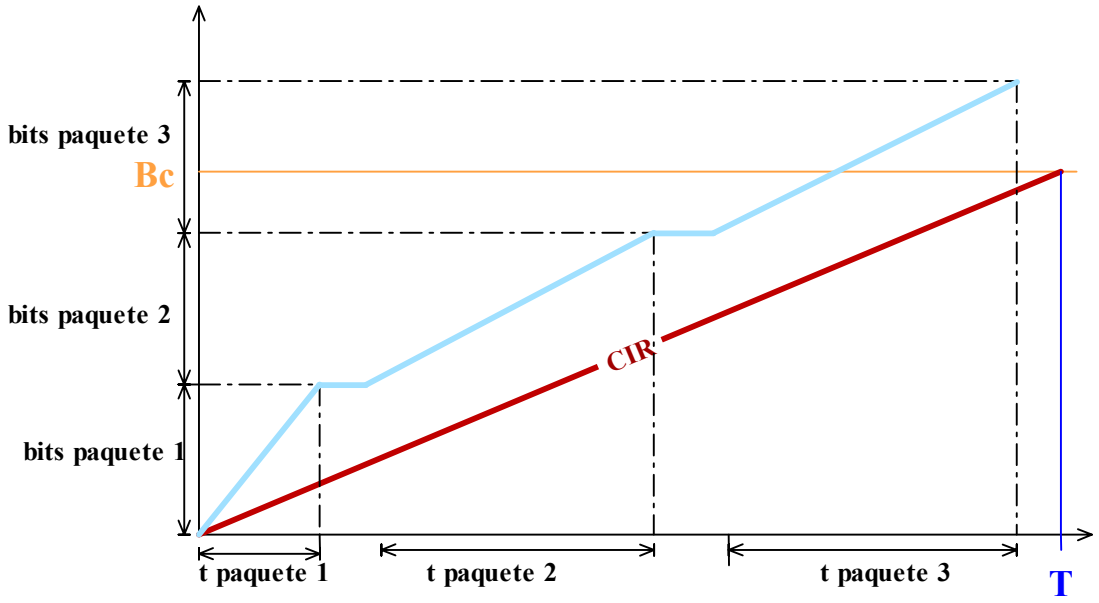

Figura 2. 21: Relación entre los parámetros de un *Token Buket*

En la metáfora del cubo de fichas (*Token Bucket*), las fichas (*tokens*) son colocadas en el cubo a una cierta tasa. El cubo tiene una capacidad determinada. Si el cubo se llena, las fichas nuevas que llegan son tiradas. Cada ficha es un permiso para que la fuente pueda enviar un cierto número de bits a la red. Para transmitir un paquete, el regulador debe borrar del cubo un numero de fichas igual al en representación al tamaño del paquete. Si no hay suficientes fichas en el cubo para enviar un paquete pueden ocurrir dos cosas; que el paquete espere hasta que el cubo tenga suficientes fichas o que el paquete se descarte. Si el cubo está lleno de fichas, las fichas que lleguen desbordarán y no estarán disponibles para futuros paquetes. Así, una gran ráfaga puede ser enviada a la red si es aproximadamente proporcional al tamaño del cubo.

Existe un algoritmo especial de *Token Bucket* con doble *Token Bucket*. En este tipo de algoritmos aparece un cuarto término que será el **Be (Excess Burst Size ó Exceso de tráfico)** que es la cantidad de bits transmitidos en el periodo T por encima de la tasa CIR.

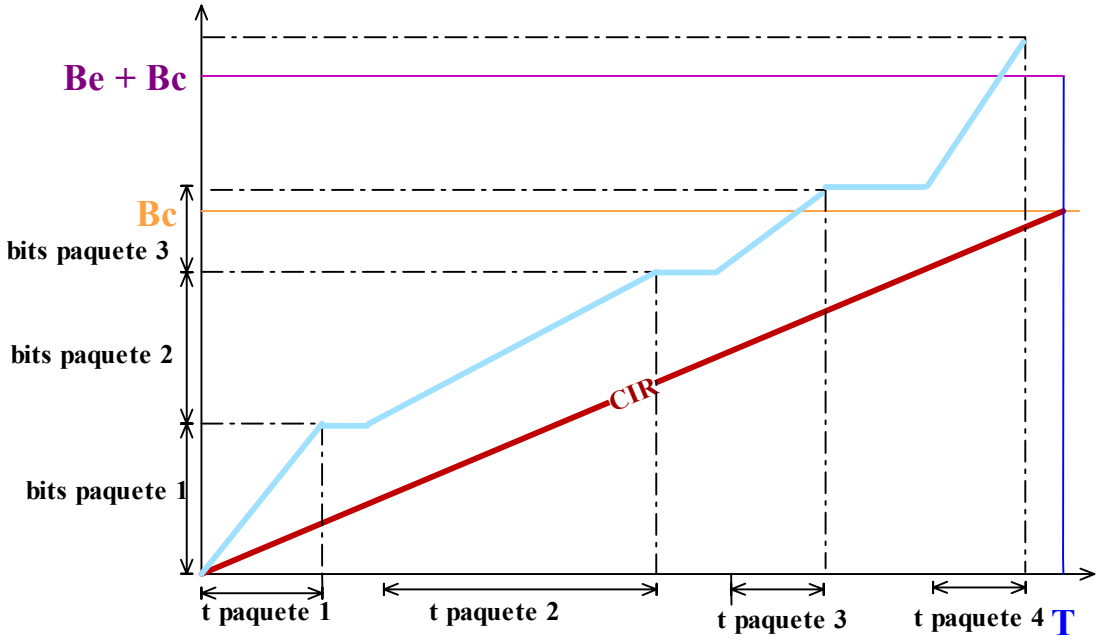

Figura 2. 22: Relación entre los parámetros de un *Token Buket* con doble *Token Bucket*

En la Figura 2. 22, los paquetes 1 y 2 estarán conformes con la tasa *CIR*, el paquete 3 sobrepasará la tasa *CIR* pero caerá en los límites del tráfico en exceso (puede ser marcado, descartado o encolado) y el paquete 4 sobrepasará ambos límites y lo más probable es que sea descartado. Dependiendo de la política del *Token Bucket*, los paquetes se podrían marcar con tres colores diferentes (uno para los conformes, otro para los que exceden y otro para los que sobrepasan ambos límites) o se podrían marcar sólo con dos colores y descartar aquellos paquetes que exceden todos los límites, etc.

2.5 Requisitos que impone *DiffServ* a la red

Para proporcionar un buen servicio diferenciado extremo a extremo, todos los nodos a lo largo del camino deberán ser *DS-compliants* de lo contrario no se podrá asegurar que los paquetes reciban el servicio requerido, ya que al atravesar un nodo *non-DS-compliant* éste puede dar un tratamiento impredecible al flujo de datos produciéndose pérdidas, retrasos, etc. Por esta razón, es necesario que las redes que vayan a implementar los Servicios Diferenciados sigan la arquitectura explicada en este documento, sacada directamente del *rfc2475* [5] creado por el *IETF*. Como recordatorio, los nodos que implementan los Servicios Diferenciados se unen entre sí formando dominios *DS*. Dentro de estos dominios podemos encontrar fundamentalmente dos tipos de nodos *DS*; los nodos interiores, cuya función está limitada a encaminar paquetes aplicándoles ciertas políticas según el valor del campo *DSCP* del paquete, aunque en ocasiones también hagan algo de remarcado, y los nodos frontera del dominio que serán los encargados de realizar una clasificación de los paquetes para poder mapearlos a un valor de *DSCP* oportuno antes de que los paquetes entren al dominio y la función inversa al salir del mismo.

También debemos tener en cuenta a la hora de diseñar un dominio *DS* que éste va a comunicarse con otros dominios que pueden ser *DS* o no, por lo tanto, deberán establecerse acuerdos entre los diferentes dominios sobre cómo se deben enviar los flujos de datos unos a otros. Por ejemplo, el tráfico que abandone un dominio *DS* que está conectado a un dominio que no entiende el campo *DSCP* será enviado de forma que este último entienda el flujo de tráfico

que le llega, para ello habrán tenido que ponerse de acuerdo los administradores de la red con anterioridad.

2.6 Problemas en la implementación de *DiffServ* en las redes actuales

Los problemas en la implementación de *DiffServ* en las redes actuales vienen derivados principalmente de los requisitos impuestos a la red anteriormente comentados y de la interoperabilidad con los nodos *non-DS-compliant*.

Nosotros definimos un nodo *non-differentited services-compliant* como un nodo que no interpreta el campo *DS* como viene especificado por el estándar y/o no implementa alguno o todos los *PHBs* estandarizados. Esto puede deberse a las capacidades de configuración del nodo.

Se define un *Legacy Node* como un caso especial de nodo *non-DS-compliant* que implementa la clasificación y el encaminamiento según *IPv4 precedence* como viene definido en los *rfcs 791* [3] y *1812* [17], pero que por otro lado no es *DS-compliant*. Los valores de precedencia en el octeto *ToS* de *IPv4* son compatibles con los *Class Selector Codepoints* y el comportamiento de encaminado precedente *(the precedence forwarding behavior)* cumple con los requerimientos del *Class Selector PHB*. Una distinción clave entre un *Legacy Node* y un nodo *DS-compliant* es que un *Legacy Node* puede interpretar o no los bits del 3-6 del octeto *ToS*. Nodos que son *non-DS-compliants* y no son *Legacy Nodes* pueden ocasionar un encaminamiento impredecible para paquetes que no tengan sus *DS codepoint* a cero.

Los Servicios Diferenciados dependen de los mecanismos de reserva de recursos ofrecidos por la implementación de los *PHBs* en los nodos. Las garantías del nivel de calidad de servicio pueden venirse a bajo en el caso de que el tráfico transite por un nodo *non-DS-compliant*, o un dominio *non-DS-capable*.

Podemos examinar dos casos separados. El primer caso es debido al uso de nodos *non-DS-compliants* dentro de un domino *DS*. La falta de encaminamiento *PHB* en un nodo puede hacer imposible ofrecer bajo retardo, pocas perdidas o servicios de aprovisionamiento de ancho de banda a través del camino en el que está situado el nodo. Sin embargo, el uso de *Legacy Nodes* puede ser una alternativa aceptable, asumiendo que el domino *DS* use solamente los *Class Selector Codepoints*, y asumiendo también que la particular implementación de la precedencia en el *Legacy Node* ofrece un encaminamiento compatible con los servicios ofrecidos a lo largo de los caminos a los que pertenece el nodo. Es importante restringir el uso de los codepoints a los *Class Selector Codepoints*, ya que los *Legacy Nodes* pueden no interpretar los bits del 3-5 y se pueden producir situaciones impredecibles.

El segundo caso es respecto al paso de los flujos de tráfico que reciben un servicio a través de dominios *non-DS-capable*. Asumimos como motivo de la discusión que un dominio *non-DS-capable* no realiza acondicionamiento del tráfico en los nodos frontera; por lo tanto, incluso en el caso que el dominio consista en *Legacy Nodes* como nodos interiores, la falta de acondicionado del trafico en los nodos frontera limitará la habilidad de proporcionar algunos tipos de servicios de modo consistente a través del dominio. Un dominio *DS* y un dominio *non-DS-capable* pueden negociar un acuerdo que decidirá como el tráfico de salida del dominio *DS* debe ser marcado antes de entrar dentro de un domino *non-DS-capable*. Alternativamente, cuando hay conocimiento de que el dominio conectado está formado por *Legacy Nodes*, el flujo de subida de un domino *DS* puede re-marcar oportunamente el tráfico de los Servicios Diferenciados a uno o más *Class Selector codepoints*. Cuando no se conozcan las características del dominio conectado se pueden marcar los paquetes con el *DSCP* a cero, asumiendo que los nodos de ese dominio darán a ese nodo el servicio de *Best Effort*.

En el caso de que el flujo de datos vaya en el otro sentido, el tráfico será acondicionado en los nodos de ingreso del domino *DS*.

En definitiva, hay muchas redes con nodos o *routers* cuya tecnología no les permite implementar los Servicios Diferenciados. Muchos de los *routers* actuales dan la opción de implementar los Servicios Diferenciados si se desea, pero los *routers* menos recientes no incluyen esta posibilidad y muchos proveedores de servicio no están dispuestos a asumir la gran inversión que supone cambiar toda la red para que la arquitectura de *DiffServ* funcione correctamente. Y como se ha expuesto, mezclar nodos *DS* con nodos *non-DS-capables* da resultados impredecibles.

2.7 Soporte de *DiffServ* por parte de los fabricantes

Los fabricantes principales de *routers*, actualmente, incluyen en sus equipos la posibilidad de implementar los Servicios Diferenciados. Para ello, pueden crear como Cisco [18], sus propias herramientas de configuración y añadir diversas funcionalidades adicionales a sus equipos.

Uno de los motivos por el que se ha optado por el equipamiento Cisco para la realización de este proyecto es que Cisco *Systems* es el líder mundial en conexión de redes de Internet. La mayoría de las redes corporativas, de educación y gubernamentales alrededor del mundo emplean las soluciones de interconexión de redes basadas en el protocolo *IP* de Cisco. Cisco proporciona una línea extensa de soluciones para transportar datos, voz y video dentro de edificaciones, campus o alrededor del mundo.

Actualmente, Internet y la conexión de ordenadores son partes esenciales de las comunicaciones comerciales, de enseñanza y personales y del ocio. Virtualmente todos los mensajes o transacciones que pasan por Internet se llevan rápidamente y de forma segura a través de los equipos de Cisco. Las soluciones de Cisco aseguran que las redes públicas y privadas operan con la seguridad y flexibilidad máximas. Además, las soluciones de Cisco son la base de la mayoría de las redes grandes y complejas empleadas por las corporaciones, instituciones públicas, compañías de telecomunicación y se encuentran en crecimiento en redes de empresas comerciales de tamaño medio.

"Cisco *Internetworking Operating System"* (Cisco *IOS*) [19] es el software de redes líder de la industria y más extensamente desplegado. Proporciona servicios de red inteligentes en una infraestructura flexible de interconexión que permite un despliegue rápido de las aplicaciones de Internet.

Cisco *IOS* software contiene actualmente las herramientas necesarias para implementar los Servicios Diferenciados. Además, este software está en continua actualización y los *routers* pueden cambiar fácilmente de *IOS*. Esto permitirá agregar nuevas funcionalidades a los *routers* conforme vayan apareciendo, adaptándose al entorno sin quedar obsoletos.

Para implementar los Servicios Diferenciados el software Cisco *IOS* dispone de herramientas como el *Modular Quality of Service Command-Line Interface (MQC)* que mediante un entorno de interfaz de comandos permite configurar el resto de herramientas que Cisco emplea para implementar los Servicios Diferenciados, tales como *Class-Based Packet Marking, Class-Based Policing, Class-Based Shaping, WRED, CBWFQ, LLQ,* etc, que se verá en profundidad en el siguiente capítulo.

Capítulo 3
Servicios Diferenciados en Cisco

3.1 Introducción

Este capítulo se divide fundamentalmente en dos grandes bloques. El primero de ellos titulado 'El software Cisco *IOS*' muestra conceptos generales sobre el software de redes que utilizan los routers. En él se verá una pequeña introducción de este sistema operativo tan extesamente desplegado en las redes actuales así como de su proceso de evolución y lanzamiento al mercado. Por otro lado, se verá cómo utilizarlo para configurar el *router* y cómo actualizarlo con las nuevas versiones que aparezcan.

El segundo gran bloque, títulado 'Los Servicios Diferenciados en el Software de Cisco' muestra las herramientas incluidas en el software Cisco *IOS* empleadas para implementar la arquitectura de Servicios Diferenciados. En primer lugar, se hará un breve resumen de los componentes de la arquitectura de Servicios Diferenciados y se verá la implementación que Cisco hace de cada uno a grandes rasgos. En segundo lugar se explicará el *MQC*, herramienta que el software Cisco *IOS* proporciona para configurar los *DiffServ* de un modo sencillo mediante el uso de comandos. Finalmente, se expondrán las herramientas con las que el software Cisco *IOS* implementa cada uno de los elementos de la arquitectura de Servicios Diferenciados y que se configuran usando el *MQC*.

3.2 El Software Cisco Internetworking Operating System (*IOS*)

"Cisco Internetworking Operating System" (Cisco *IOS*) [19]es el software de redes líder de la industria y más extensamente desplegado. Proporciona servicios de red inteligentes en una infraestructura flexible de interconexión que permite un despliegue rápido de las aplicaciones de Internet.

Cisco *IOS* software está en continua expansión de sus capacidades y usos. Las necesidades del cliente dirigen las capacidades del Cisco *IOS* software, que proporciona un extenso rango de funcionalidades: desde conectividad básica, seguridad y gestión de redes hasta avanzados servicios tales como comercio en tiempo real, soporte interactivo, etc.

La funcionalidad del Cisco *IOS* software es el resultado de una evolución. La primera generación de dispositivos de interconexión sólo podían almacenar y encaminar paquetes de datos. Actualmente, Cisco *IOS* software puede reconocer, clasificar y priorizar el trafico de la red, optimizar el enrutado, soportar aplicaciones de video y de voz y muchas cosas más.

Cisco *IOS* software se ejecuta en la mayoría de routers de Cisco e incrementalmente en los switches de Cisco. Estos dispositivos de red manejan la mayor parte del tráfico de Internet actualmente.

Como se muestra en la siguiente figura, si el Cisco *IOS* software y las plataformas de red de Cisco se ejecutan unidos forman un sistema unificado, el cuál es una base sólida sobre la que construir las aplicaciones de Internet.

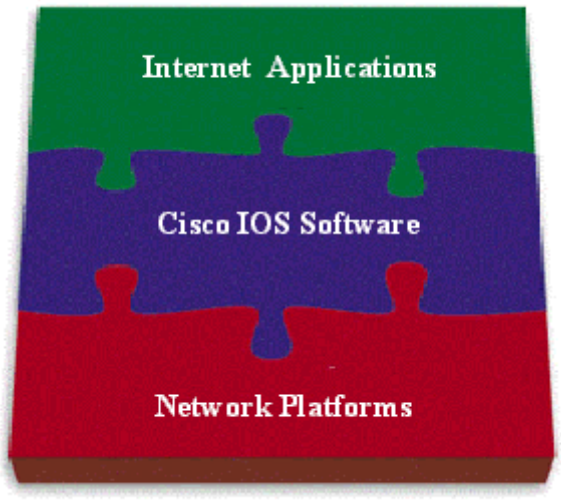

Figura 3. 1: Unión de las Plataformas de Red con el Software Cisco *IOS*

El proceso de lanzamiento al Mercado del software Cisco *IOS* tiene tres categorías: versión de instalación anticipada (ED), versión principal y versión de instalación general (GD).

Las versiones ED están diseñadas para ofrecer capacidades y características específicas y deben instalarse de forma selectiva donde sea necesario. Dado que tienen un ciclo vital limitado, deben actualizarse a una versión principal.

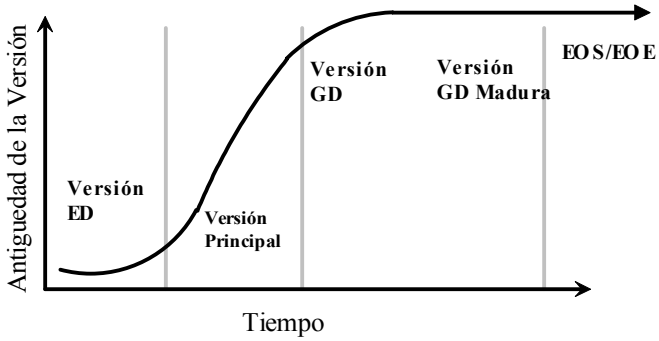

Figura 3. 2: Ciclo de vida de las versiones del software Cisco *IOS*

Cisco proporciona las siguientes versiones especiales para cubrir las necesidades específicas de clientes y empresas:

- Las versiones "X" son de ciclo de vida corto y estan diseñadas para proporcionar soporte para nuevas plataformas hardware, módulos de red o herramientas software. Las versiones "X" se consolidan normalmente dentro de una versión ED posterior.

- Las versiones S están diseñadas para las series de Cisco 12000 GSR y Cisco 7200/7500 para aplicaciones de red interiores.

- Las versiones E están diseñadas para proporcionar tecnologías de comercio críticas a redes empresariales.

El siguiente paso en el ciclo de vida del software de Cisco *IOS* es la versión principal, la cual unifica las herramientas, plataformas, funcionalidades y tecnologías de versiones más tempranas. El énfasis está en escalabilidad y confiabilidad.

Las versiones principales se encaminan a alcanzar una certificación GD que se le adjudica al Cisco *IOS* software que ha tenido una extensa exposición en el mercado en un extenso rango de entornos de red.

3.2.1 Modos de configuración del Router

La interfaz del Cisco *IOS* se divide en muchos modos diferentes. Los comandos disponibles en cada momento dependen del modo en el que se esté. Utilizando el interrogante (?) el sistema muestra la lista de comandos disponibles en cada modo de comando.

Cuando se conecta con el software de Cisco *IOS*, se comienza en el modo usuario, frecuentemente llamado modo EXEC. Para tener acceso a todos los comandos, se debe entrar en modo privilegiado. Normalmente, se debe introducir una contraseña para entrar en el modo EXEC privilegiado. Desde el modo privilegiado, puedes introducir algunos comandos EXEC o entrar en el modo de configuración global. Los comandos EXEC no se salvan cuando se rebota el sistema.

Los modos de configuración permiten realizar cambios en la *running configuration*. Si después almacenas la configuración en la *startup configuration,* estos comandos se almacenan cuando el sistema se rebota. Para introducir los diferentes comandos de configuración, se debe comenzar en el modo de configuración global. Desde el modo de configuración global, se pueden entrar en los modos de configuración de la interfaz, subinterfaz y en varios modos de configuración protocolos específicos.

Modo de ROM monitor es un modo separado empleado cuando un dispositivo de red que ejecuta el software de Cisco *IOS* no puede iniciarse de manera apropiada. Si el dispositivo de red no encuentra una imagen de sistema válida cuando se está iniciando, o si su archivo de configuración está corrupto en el *startup*, el sistema puede entrar en el modo de ROM monitor.

Tabla 3. 1: Modos de Configuración del Router

Modo Comando	Método de Acceso	Símbolo mostrado	Comando para salida
EXEC de usuario	Entrada al sistema (Log in)	`Router>`	Exit
EXEC privilegiado.	Desde el modo EXEC Introducir el comando enable	`Router#`	Exit
Configuración Global	Desde el modo Privilegiado EXEC, introducir configure terminal	`Router (config)#`	Exit
Configuración de Interfaces	Desde el modo de configuración Global, introducir el comando Interface “interfaz” Ej: interface serial 0/0	`Router (config-if)#`	Exit

3.2.2 Cómo Obtener Ayuda en el Software de Cisco *IOS*

Introduciendo el interrogante (?) el sistema muestra una lista de los comandos disponibles para cada modo de comando. Se puede obtener también una lista de palabras claves y argumentos asociados con algún comando usando la herramienta de ayuda sensible al contexto.

Para obtener ayuda específica sobre un modo de comando, un comando, una palabra clave, o un argumento se empleará uno de los siguientes comandos:

Tabla 3. 2: Comandos de Ayuda

Comando	Propósito
help	Se obtiene una pequeña descripción del sistema de ayuda en alguno de los modos de comando.
entrada-de-comando-abreviada**?**	Se obtiene una lista de los comandos que comienzan con los caracteres introducidos antes de la interrogación.
entrada-de-comando-abreviada<**Tabulador>**	El sistema completa la parte del nombre del comando que falta.
?	Lista todos los comandos disponibles del modo de comando en el que se encuentra.
Comando **?**	Lista las palabras claves o argumentos que se pueden introducir después del comando.

3.2.3 Cómo usar las Formas No y Default de los Comandos

Casi todos los comandos de configuración tienen una forma **no.** En general la forma **no** deshabilita una función. Se empleará el comando sin la palabra clave **no** para reactivar una función desactivada. Por ejemplo, *IP routing* viene activado por defecto. Para desactiva *IP routing*, usaremos el comando **no ip routing** y emplearemos el comando **ip routing**.

Los comandos de configuración también pueden tener una forma **default**. La forma **default** de un comando pone los parámetros del comando a sus valores por defecto.

3.2.4 Cómo Guardar los Cambios de Configuración

Introducir el comando **copy running-config startup-config** para salvar los cambios en la configuración en la *startup configuration* y conseguir así que no se pierdan cuando el sistema se rebote. Por ejemplo:

```
Router# copy running-config startup-config
Building configuration...
```

Puede tardar un par de minutos en guardar la configuración. Después de que se haya salvado la configuración aparece lo siguiente:

```
[OK]
Router#
```

En la mayoría de las plataformas este comando almacena la configuración en la *NVRAM*.

3.2.5 Cómo hacer Búsquedas y Filtrados de la salida de los comandos show y more.

En las versión del Cisco *IOS* 12.0(1)T y posteriores, se pueden hacer búsquedas y filtrados de la salida de los comandos **show** y **more.** Esta función se emplea mucho cuando se necesita acortar el contenido mostrado por el sistema o si se quiere evitar que el sistema muestre algo que no se necesita ver.

Para usar esta funcionalidad, se introducirá el comando **show** y **more** más el carácter tubería (|), seguido de una de las palabras clave **begin, incluye** o **exclude** y una expresión que se quiera buscar o filtrar:

command / **{ begin | incluye | exclude }** *expresión*

Ejemplo:

El ejemplo siguiente se quiere que la salida del comando **show interface** muestre solamente las lineas en las que aparezca la expresión "protocol":

```
Router# show interface | include protocol
FastEthernet0/0 is up, line protocol is up
Serial4/0 is up, line protocol is up
Serial4/1 is up, line protocol is up
Serial4/2 is administratively down, line protocol is down
Serial4/3 is administratively down, line protocol is down
```

3.2.6 Cómo Configurar el Router

Ahora se verán los modos y métodos de configuración del *router* para actualizar el archivo de configuración del mismo [20]. Los archivos de configuración del *router* se pueden originar desde la consola, desde la *NVRAM* o desde servidores *TFTP*. Un *router* utiliza la siguiente información desde el archivo de configuración cuando se inicializa:

- Versión del software de Cisco *IOS*.
- Identificación del *router*.
- Ubicaciones del archivo de arranque.
- Información de protocolo.
- Configuraciones de interfaz.

El archivo de configuración contiene comandos para personalizar el funcionamiento del *router*. El *router* utiliza esa información cuando se inicializa. Si no hay disponible ningún archivo de configuración, el diálogo de configuración del sistema le guía por el proceso de creación de uno como se veía al iniciar por vez primera el *router*.

La información de configuración del *router* se puede originar por varios medios. se puede utilizar el comando EXEC privilegiado **configure** para configurar desde un terminal virtual (remoto) hasta una conexión por módem o el terminal de consola. Esto le permite introducir cambios en cualquier momento en una configuración que ya existe.

Se puede emplear el comando EXEC privilegiado **configure** para cargar una configuración desde un servidor de red *TFTP*, que permite mantener y almacenar información de configuración en un sitio central.

La siguiente figura ilustra el resumen de comandos de configuración:

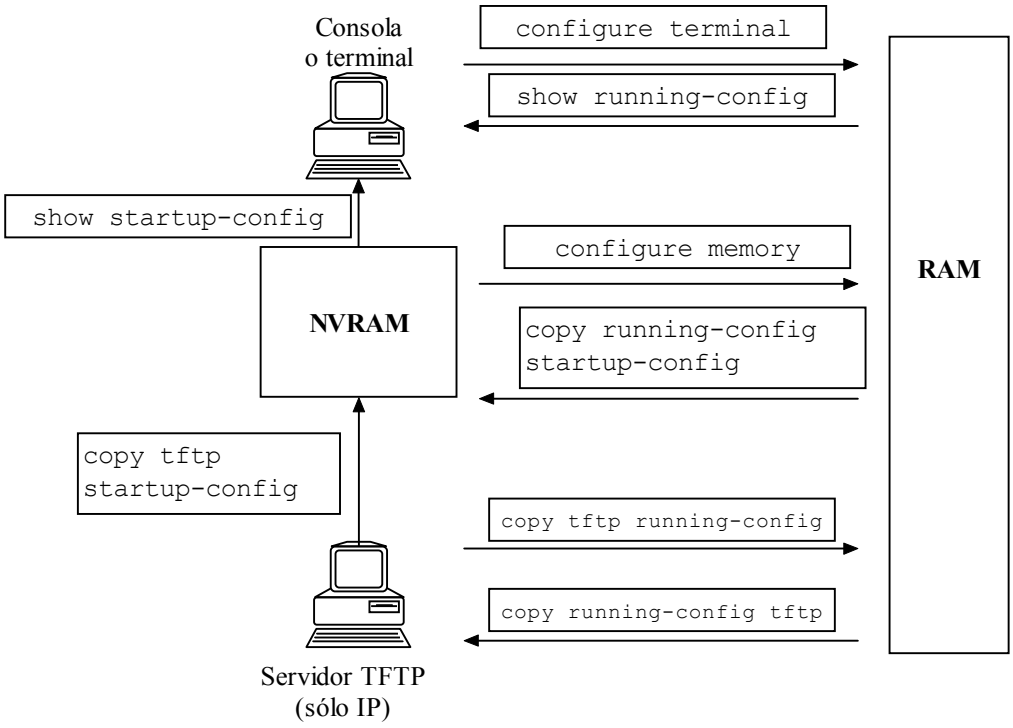

Figura 3. 3: Esquema con los comandos de configuración del *router* [20]

La siguiente tabla muestra un resumen de los comandos de configuración:

Tabla 3. 3: Comandos de configuración del *router* [20]

Comando	Descripción
configure terminal	Configura el *router* manualmente desde el terminal de la consola.
configure memory	Carga información de configuración desde la memoria de accseo aleatorio no volátil (*NVRAM*).
copy tftp running-config	Carga información de configuración desde un servidor de red *TFTP*.
show running-config	Visualiza la configuración actual almacenada en la RAM.
copy running-config startup-config	Almacena la configuración actual de la RAM en la *NVRAM*.
copy running-config tftp	Almacena la configuración actual de la RAM en un servidor *TFTP*.
show startup-config	Visualiza el configuración guardada que es el contenido de la *NVRAM*.
erase startup-config	Borra el contenido de la *NVRAM*.

3.2.7 Comandos de configuración de interfaz

Los comandos de configuración de interfaz modifican el funcionamiento de un puerto Ethernet, Token Ring o serie.

Tabla 3. 4: Comandos de configuración de Interfaz

Comando	Descripción
Router(config)#**interface** tipo-de-ranura/puerto	Especifica la interfaz que se quiere configurar. Entra en el modo de configuración de la interfaz.
Router(config-if)#**shutdown**	Se emplea para desactivar administrativamente la interfaz.
Router(config-if)#**no shutdown**	Se usa para activar una interfaz.
Router(config-if)#**exit**	Se emplea para salir del modo de configuración de interfaz actual.

3.2.7.1.1 Cómo configurar una Interfaz Serial

Antes de conectar un dispositivo a un puerto serie se debe saber:

1. El tipo del dispositivo que se está conectando:
   - Equipo terminal de datos (*Data Terminal Equipment DTE)* dispositivo que transmite y/o recibe datos a/de un DCE (p. ej., un terminal o impresora).
   - Equipo de comunicaciones de datos *(Data Communications Equipment DCE*). Es el equipo que brinda las funciones que establecen, mantienen y finalizan una conexión de transmisión de datos (como, por ejemplo, un módem).

2. El tipo de conector, macho o hembra, necesario para conectar al dispositivo.

3. La señalización estándar requerida por el dispositivo.

Un dispositivo DCE proporciona una señal de reloj que temporiza las comunicaciones entre el dispositivo y el *router*. Un dispositivo DTE no proporciona la señal de reloj. Normalmente el DTE suele ser llevar un conector macho y un DCE hembra.

En enlaces serie, el extremo DCE debe proporcionar una señal de temporización; el otro lado es un DTE. Por defecto, los routers Cisco son dispositivos DTE; sin embargo, en algunos casos se pueden utilizar como dispositivos DCE.

Si se va a utilizar la interfaz para proporcionar temporización, se debe especificar una velocidad con el comando **clockrate**. El comando **bandwidth** sobrescribe el ancho de banda predeterminado que se visualiza con el comando **show interface** y es utilizado por varios protocolos de enrutamiento.

Todas las señales serial están sujetas a limitaciones por la distancia. Normalmente, disminuye la tasa de transferencia conforma aumenta la distancia. Por ejemplo, la tasa máxima recomendada para V.35 es 2 Mbps, pero comunmente se utilizan los 4 Mbps.

3.2.8 Imágenes del Software Cisco *IOS* [21]

Como se ha expuesto, hay varias versiones del software de Cisco *IOS* y continuamente aparecen nuevas versiones que añaden funcionalidad a las anteriores. Por lo tanto, es muy importante saber cómo se actualizan las imágenes del software de Cisco *IOS* en los routers. En este proyecto, por ejemplo, se deberán actualizar las imágenes del software de Cisco *IOS* por

versiones que dispongan de las herramientas adecuadas para la implementación de los Servicios Diferenciados.

Los routers inician el software de Cisco *IOS* desde los siguientes orígenes:
- Memoria *flash*
- Servidor *TFTP*.
- ROM (no el sofware de Cisco IOS completo).

3.2.9 Localización del software de Cisco *IOS*

El origen predeterminado del arranque del software de Cisco *IOS* depende de la plataforma hardware, pero lo más habitual es que el *router* mire los comandos de arranque del sistema guardados en la *NVRAM*. Sin embargo, el software de Cisco *IOS* permite utilizar varias alternativas. Se pueden especificar otras fuentes para que el *router* busque el software.

Como se describe en la siguiente figura, los parámetros predeterminados en el registro de configuración permiten las siguientes alternativas: puede especificar comandos de arranque del sistema en modo de configuración global que el *router* empleará al reiniciar para saber de dónde cargar la *IOS*. Estas sentencias se guardarán en la *NVRAM* para utilizarlas durante el siguiente arranque utilizando el comando **copy running-config startup-config**. Si la *NVRAM* carece de comandos de arranque del sistema que el *router* pueda utilizar, el sistema tiene su propia secuencia de *fallback*. Por defecto, el *router* puede utilizar Cisco *IOS* en la memoria *flash*. Si la memoria *flash* está vacía, el *router* puede intentar su siguiente alternativa: *TFTP*.

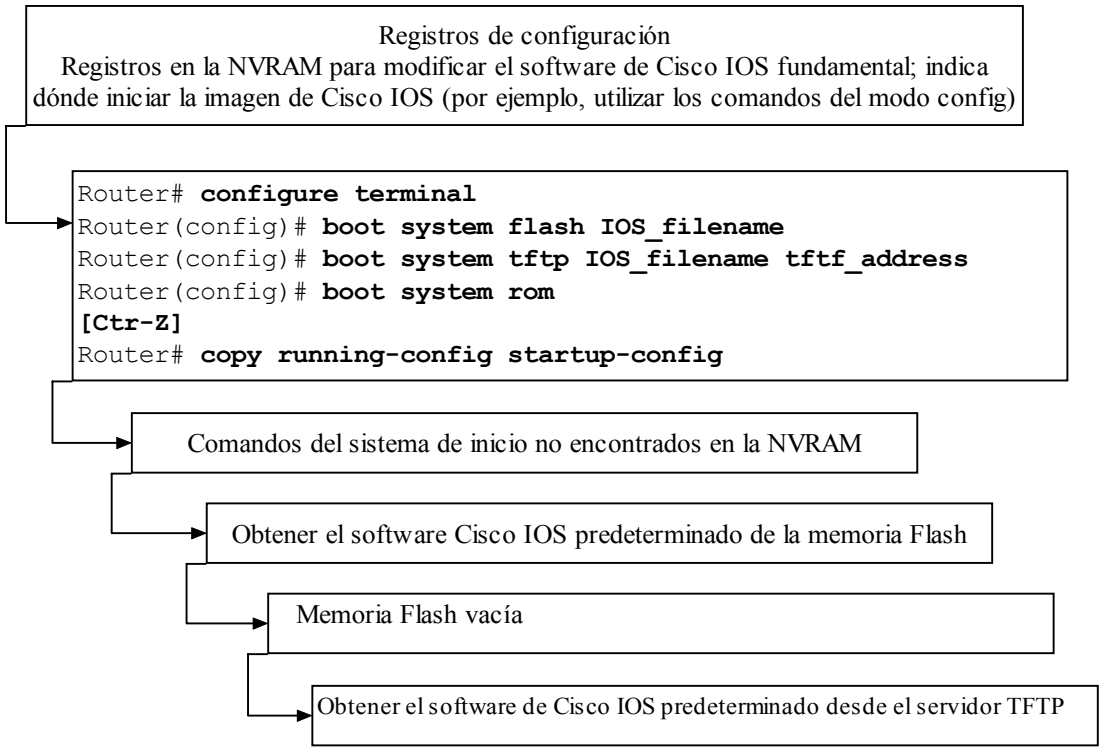

Figura 3. 4: Los ajustes en el registro de configuración permiten alternativas acerca de dónde el *router* buscará el software de Cisco *IOS*. [21]

3.2.10 Valores del registro de configuración

El orden en el que el *router* busca las imágenes de *IOS* para cargarlas depende del parámetro del campo boot en el registro de configuración. Se puede cambiar el parámetro predeterminado del registro de configuración con el comando de configuración global **config-register**. Se empleará un número hexadecimal como argumento para este comando en el siguiente ejemplo:

```
Router# configure terminal
Router(config)#config-register 0x10F
```

En este ejemplo, el registro de configuración está configurado de tal manera que el *router* examinará el archivo de inicio que hay en la *NVRAM* para buscar las opciones de arranque del sistema. El registro de configuraciones es un registro de 16 bits que está en la *NVRAM*. Los cuatro bits más bajos del registro de configuración (bits 3,2,1 y 0) forman el campo boot. Para cambiar el campo boot y dejar los demás bits configurados a sus valores originales, se seguirán estas pautas:
Configuarar el valor del registro de configuración a 0x100 si se necesita entrar en el monitor ROM (principalmente en un entorno de programador). Desde este monitor, arranque el sistema operativo manualmente utilizando el comando b en la línea de comandos del monitor ROM. (Esto configura los bits del campo boot a 0-0-0-0.)

Configurar el registro de configuración a 0x101 para hacer que el sistema arranque automáticamente desde la ROM. (Esto configura el campo boot a 0-0-0-1.)

Configurar el registro de configuración a cualquier valor desde 0x102 a 0x10F para hacer que el sistema utilice los comandos de arranque del sistema de la *NVRAM*. Esto es lo predeterminado. (Esto configura los bits del campo boot entre 0-0-1-0 y 1-1-1-1.)

Para comprobar el parámetro del campo boot y para verificar el comando **config-register**, se empleará el comando **show versión**.
El comando **show versión** visualiza información sobre la versión software Ciso *IOS* que se está ejecutando actualmente en el *router*. Esto incluye el registro de configuración y el ajuste del campo boot.

3.2.11 Opciones de bootstrap en el software

Se pueden introducir múltiples comandos de arranque del sistema para especificar la secuencia que seguirá el sistema a la hora de arrancar el software de Cisco *IOS*.

Memoria flash: se puede cargar una imgen del sistema desde la memoria de sólo lectura programable y borrable eléctricamente (EEPROM). La ventaja es que la información almacenada en la memoria Flash no es vulnerable a los fallos de la red que pueden ocurrir cuando se cargan imágenes desde servidores *TFTP*:

```
Router# configure terminal
Router(config)#boot system flash gsnew-image

Router#copy running-config startup-config
```

***TFTP*:** Si la memoria Flash está dañada se puede cargar la imgen desde un servidor de red.

```
Router# configure terminal
Router(config)#boot system tftp test exe 172.16.13.11

Router#copy running-config startup-config
```

ROM: Si la memoria Flash está dañada y el servidor de red falla al cargar la imagen, arrancar desde la ROM será la opción final de arranque del software. Sin embargo, la imagen del sistema que hay en la ROM es un subconjunto del software Cisco *IOS* que carece de los protocolos, elementos y configuraciones del software Cisco *IOS* completo. Asimismo, si se ha actualizado el software desde que se adquirió el *router*, es probable que se trate de una versión más antigua del software Cisco *IOS*.

```
Router# configure terminal
Router(config)#boot system rom

Router#copy running-config startup-config
```

El commando **copy running-config startup-config** guarda los commandos en la *NVRAM*. El *router* ejecutará los comandos de arranque del sistema según los necesite en el orden en el que se introdujeron originalmente en el modo de configuración.

3.2.12 Preparación para el uso de *TFTP*

Las internetworks de producción habitualmente abarcan amplias áreas y contienen múltiples routers. Estos routers geográficamente distribuidos necesitan una localización de seguridad para las imágenes del software. Un servidor *TFTP* habilita a la imagen y a la configuración a que se carguen y descarguen en la red. El servidor *TFTP* puede ser otro *router* o un sistema *host*. El *host TFTP* puede ser cualquier sistema que tenga software de servidor *TFTP* cargado y que funcione y que sea capaz de recibir archivos desde la red *TCP*/*IP*. Habrá que copiar el software entre el servidor *TFTP* y la memoria Flash del *router*. Sin embargo, antes de hacer esto, se deben comprobar las siguientes condiciones preliminares:

Desde el *router* hay que verificar que se puede acceder al servidor *TFTP* sobre la red *TCP*/*IP*. El comando ping es un método que puede ayudar a hacer esto:

```
Router#ping tftp-address
!!!!! (hay respuesta)
```

Verificar que el *router* tiene suficiente espacio en la memoria Flash para acomodar la imagen del software de Cisco *IOS*:

```
Router# show flash
4096 kbytes of flash memory on embedded flash (in XX).
file         offset       length       name
 0           0x40         1204637      xk09140z
[903848/2097152     bytes  free]
```

En el servidor *TFTP* hay que asegurarse de conocer el nombre del archivo y la ubicación de la imagen de l software de Cisco *IOS*.

3.2.13 Creación de una copia de seguridad de la imagen del software

Antes que nada, se debería hacer una copia de la imagen actual del sistema en un servidor de red *TFTP*.

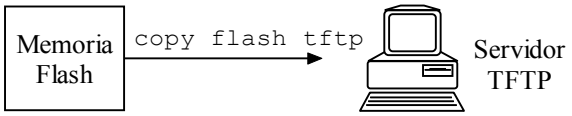

Figura 3. 5: Copia de la imagen del software Cisco *IOS* de la memoria flash al servidor *TFTP*

Los pasos a seguir son:

Tabla 3. 5: Pasos a seguir para copiar la imagen del software de Cisco *IOS* de la memoria flash al servidor *TFTP*

	Comando	Descripción
Paso1:	`Router#`**`show flash`**	Muestra datos sobre la versión actual de la imagen del software Cisco *IOS* de nuestro sistema. Necesario para saber el nombre del archivo de imagen.
Paso2:	`Router#`**`copy flash tftp`**	Copiar el contenido de la memoria flash en un servidor *TFTP*.
Paso3:	*`IP`* `adress of remote` *`host`*`[255.255.255.255]?` **`dirección-IP-del-servidor`**	Dirección del servidor *TFTP*.
Paso4:	`Filename to write on tftp hose?` **`Nombre-del-archivo`**	Nombre del archivo de imagen que se quiere copiar.

3.2.14 Cargar nueva imagen desde servidor *TFTP*

Una vez hecha la copia de seguridad de la imagen actual del software de Cisco *IOS*, se puede cargar una nueva imagen.

Figura 3. 6: Copia de la imagen del software de Cisco *IOS* del servidor *TFTP* a la memoria flash

Para ello se emplerá el comando **copy tftp flash**. Los pasos a seguir son:

Tabla 3. 6: Pasos para copiar la imagen del software de Cisco *IOS* del servidor *TFTP* a la memoria flash

	Comando	Descripción
Paso1:	`Router#`**`copy tftp flash`**	Copiar el contenido de un servidor *TFTP* dentro de la memoria Flash.
Paso2:	*`IP`* `address or name of remote` *`host`*`[255.255.255.255]?` **`dirección-IP-del-servidor`**	Dirección del servidor *TFTP*.
Paso3:	`Name of tftp filename to copy into flash []?`**`Nombre-del-archivo-de-imagen`**	Nombre del archivo de imagen que se quiere copiar.
Paso4:	`Confirmación? <Return>`	Confirmación.

	Comando	Descripción
Paso5:	`Erase flash before writing?[confirm]<Return>`	Si no hay memoria suficiente para las dos imágenes. El sistema pregunta si se quiere borrar el contenido de la memoria flash antes de realizar la copia.

Utilizar el comando **show versión** para comprobar que el proceso se ha realizado satisfactoriamente.

3.3 Los Servicios Diferenciados en el Software de Cisco

En el apartado 3.2 se ha hablado del software Cisco *IOS* que, como se ha visto, es el software que se emplea para controlar y configurar las distintas funcionalidades del *router*. En ese apartado se han expuesto los conceptos básicos y previos para la utilización de ese software de red en los routers Cisco, así como la manera de actualizarlo con las nuevas versiones. En este apartado se hará un breve resumen de los componentes que forman parte de la arquitectura de Servicios Diferenciados y se expondrán las herramientas disponibles en software de Cisco *IOS* para implementarlos. En los apartados posteriores, se explicarán con más detalle algunas de estas herramientas, en concreto, aquellas que se han usado en el proyecto para implementar los *DiffServ*.

Actualmente el modelo de Servicios Diferenciados sólo define el uso del *DSCP* y de los cuatro *PHBs*. Los *PHBs* simplemente describen la forma de encaminamiento de un nodo *DS-compliant*. El modelo no especifica cómo serán implementados estos *PHBs*. Una variedad de técnicas de encolado (*queueing),* de medida *(metering),* de espaciado *(shaping)* y funciones policía *(policing)* pueden crear los *traffic conditioning* deseados y *PHB*.

Resumen de los componentes que forman los Servicios Diferenciados [13][22]:

- **Acondicionado del Tráfico *(Traffic conditioning):* funciones policía y de espaciado *(traffic policing* y *traffic shaping)*** [39]**.** El acondicionamiento del tráfico se ejecuta en las fronteras de un dominio *DiffServ.* Para asegurar, que el tráfico que entra a un dominio *DS* respeta las reglas especificadas por el Acuerdo de Acondicionado del Tráfico *(TCA),* los *Traffic Conditioners* realizan funciones de *traffic shaping* y *policing.* Para implementar el *Traffic Conditioning* Cisco proporciona las siguientes herramientas:
  - *Traffic Policing* [16]*: Commited Access Rate (CAR), Class-Based Policing.*
  - *Traffic Shaping: Generic Traffic Shaping (GTS)* [37]*, Frame Relay Traffic Shaping (FRTS)* [38]*, Class-Based Shaping* [28]*.*

- **Clasificación de Paquetes *(Packet classification)*** [26]**.** La clasificación de paquetes usa un descriptor (por ejemplo, el *DSCP*) para clasificar un paquete dentro de un grupo específico y definir así ese paquete. Una vez clasificado, el paquete ya es accesible para los dispositivos que gestionan la *QoS* a lo largo de la red. Así, mediante la clasificación de paquetes, el tráfico de la red se puede dividir en múltiples niveles o clases de servicio.

  Para realizar la clasificación de paquetes Cisco emplea:
  - Los **class-maps** dentro del *Modular Quality of Service Command Line Interfaz MQC,* que se explicarán en detalle posteriormente.

- Las **listas de acceso** *(Acces List)*. Estas listas se explicarán en el anexo A. Con ellas se puede dividir el tráfico entrante tomando datos de la cabecera de los paquetes tales como el puerto origen o destino, la dirección *IP* de origen o destino, etc.
- Los ***Traffic Conditioners,*** que también pueden realizar clasificaciones del tráfico según se adapte o no al contrato.

- **Marcado de Paquetes *(Packet Marking)*** [27]**.** El marcado de paquetes está relacionado con la clasificación de paquetes ya que dependiendo de cómo se marque un paquete éste irá a parar a una clase o a otra. Permite clasificar un paquete basándose en un descriptor de tráfico específico (como el valor *DSCP*). Se puede usar esta clasificación para aplicar Servicios Diferenciados definidos por el usuario a los paquetes y para asociar estos paquetes con una política local de *QoS***.**

  - Cisco dispone de la herramienta ***Class-Based Packet Marking*** para realizar el marcado de los paquetes.
  - Los paquetes podrán, también, ser marcados o remarcados por un ***Traffic Conditioner***.

- **Gestión de la Congestión *(Congestion Management)*** [24]**.** Las herramientas para la gestión de la congestión controlan ésta una vez que ocurre. Una forma de que los elementos de la red manejen un desbordamiento de tráfico de entrada consiste en usar un algoritmo de encolamiento para parar el tráfico y determinar entonces algún método que establezca prioridades a la hora de usar el enlace de salida.

  Para gestionar la congestión, Cisco utiliza diversos métodos de encolamiento. Las herramienta que usa Cisco con este fin en los Servicios Diferenciados son [40]:
  - *Class-Based Weighted Fair Queueing (CBWFQ)* [29][30].
  - *Low Latency Queueing (LLQ)* [32].

- **Evitar la Congestión *(Congestion Avoidance)*** [25]**.** Estas técnicas monitorizan las cargas de tráfico de la red en un esfuerzo por anticipar y evitar la congestión en los cuellos de botella comunes de una red, antes de que se conviertan en un problema. Estas técnicas están diseñadas para dar un tratamiento preferente al tráfico de la clase *premium* (prioritaria) en situaciones de congestión mientras que al mismo tiempo maximizan el *throughput* y la capacidad de uso y minimizan las pérdidas de paquetes y el retardo.

  La herramienta que utiliza Cisco para evitar que se produzca congestión es:

  - El *Diffserv Compliant Weighted Random Early Detection (WRED)* [30][31].

3.4 Herramientas de Cisco para Implementar los Diffserv

En el apartado 3.3 se han visto a grandes rasgos las herramientas disponibles en el software de Cisco *IOS* para implementar los elementos de la arquitectura de Servicios Diferenciados. En este apartado se verán en detalle algúnas de esas herramientas. En primer lugar se hará un pequeño resúmen introductorio de cada una de ellas para tener una visión global de la explicación posterior más detallada. Las herramientas disponibles dentro del software Cisco *IOS* para implementar los Servicios Diferenciados son [13][22]:

- ***Modular QoS Command-Line Interface (CLI)*** [23]***:*** Proporciona una estructura CLI para poder configurar las diferentes herramientas de *QoS* basada en clases.

- ***Class-Based Packet Marking*** [26]***:*** Esta característica proporciona un modo fácil mediante CLI para un marcado eficiente de los paquetes. Marca los paquetes de una determinada clase definida previamente por el usuario mediante el *MQC*.

- ***Class-Based Traffic Policing*** [16]***:*** Esta característica permite limitar la tasa de transmisión de una clase de tráfico basándose en criterios definidos por el usuario. También permite al sistema marcar los paquetes con el *IP DSCP* o el *IP* Precedence.

- ***Class-Based Shaping*** [28]***:*** Permite configurar *GTS* sobre una clase de tráfico, especificando la tasa media del traffic *shaping* y configurar *CBWFQ* dentro de *GTS*.

- ***CBWFQ*** [29][30]***:*** Es un mecanismo de programación utilizado para proporcionar un mínimo de ancho de banda garantizado a clases de tráfico durante tiempos de congestión de la red.

- ***Diffserv Compliant WRED*** [30][31]***:*** Proporciona soporte para los Servicios Diferenciados estándar. Activa *WRED* para usar los valores de *DSCP* o *IP Precedence* cuando calcula la probabilidad de tirar un paquete.

- ***LLQ*** [32]***:*** La herramienta de encolamiento de baja latencia *(Low Latency Queueing)* lleva la disciplina de servicio de colas encolamiento de prioridad estricta *(Strict Priority Queueing)* a *CBWFQ*. Se emplea para dar soporte a la clase *Expedited Forwarding EF* de los Servicios Diferenciados ya que esta clase tiene necesidades de baja latencia que *LLQ* soluciona.

- ***CEF*** [33]***:*** *Cisco express Forwarding (CEF)* es una avanzada tecnología de conmutación de la capa *IP*. *CEF* optimiza el rendimiento y la escalabilidad de la red en redes con configuraciones extensas y dinámicas, tales como *Internet*. Algunas de las herramientas de Cisco para la implementación de los Servicios Diferenciados necesitan que *CEF* esté activado para funcionar.

Todas estas herramientas se configurán de forma sencilla usando la primera de ellas, el *MQC* a excepción de *CEF* que se configura aparte. En la siguiente tabla se muestra un resumen de las herramientas que utiliza Cisco para implementar los Servicios Diferenciados [13]:

Tabla 3. 7: Herramientas de Cisco para Implementar los *DiffServ*

Clasificador	Acondicionador	Encaminamiento	*PHB*	Accounting
En el *MQC*, la clasificación Está basada en: a. Interfaz b. Dirección *MAC* *c. ACL* *d. NBAR* e. Valor de *DSCP*/*IP Precedence*	*Class Based Weighted Fair Queueing (CBWFQ)* dentro del *MQC* y las herramientas de abajo, en un sistema de clases o fuera del modelo basado en clase.	*Cisco Express Forwarding (CEF)*	*Class Based Weighted Fair Queueing (CBWFQ)* dentro del *MQC* y las herramientas de abajo, en un sistema de clases o fuera del modelo basado en clase.	*Class-based accounting MIB y CLI*
QPPB	*LLQ y dLLQ*	*Policy-Based Routing (PBR)*	*LLQ y dLLQ*	

Clasificador	Acondicionador	Encaminamiento	*PHB*	Accounting
Class-Based Traffic Policing *Policy-Based Routing (PBR)*	*WRED y dWRED* *Class-Based Traffic Policing* *Class-Based Traffic Shaping* *Policy-Based Routing (PBR)*		*WRED y dWRED* *Class-Based Traffic Policing* *Class-Based Traffic Shaping* *Policy-Based Routing (PBR)*	

En definitiva, lo que debe quedar claro después de este apartado introductorio es que el software de Cisco *IOS* dispone de varias herramientas para implementar los elementos de la arquitectura de Servicios Diferenciados y entre ellas el *MQC*, que es una utilidad del software Cisco *IOS* que permite configurar de modo sencillo, mediante el uso de una interfaz de comandos, todas las demás herramientas disponibles en el *IOS* que implementan los *DiffServ* (basadas en clases).

3.4.1 Modular *QoS* CLI

El *Modular QoS CLI* [23] proporciona una estructura de interfaz de comandos para configurar las diferentes herramientas de calidad de servicio basada en clases *(QoS class-based)*. Así, *MQC* engloba al resto de herramientas de que dispone Cisco para proporcionar los Servicios Diferenciados de una forma sencilla.

Permite la separación entre la clasificación de los paquetes *(class-maps)*, las políticas *(policy-maps)* aplicadas en las clases definidas y asociar esas políticas a las interfaces correspondientes *(service-policy)*. El *MQC* forma la base para proporcionar Servicios Diferenciados y todos los mecanismos de *QoS* son parte de los *class-maps* (clasificación) o los *policy-maps* (dentro de los policy-maps se incluyen: funciones policía, espaciado, encolado, técnicas para evitar la congestión, marcado de paquetes, marcado de los bits de Clase de Servicio *CoS* de la capa de enlace de datos, etc).

Como se ha expuesto anteriormente los tres pasos que seguiríamos con el *MQC* para poder implementar los Servicios Diferenciados en el *router* serían los siguientes:

1. Definir una clase de tráfico *(traffic class)* con el comando **class-map.**
2. Crear una *service policy* asociando la clase de tráfico creada anteriormente con una o más políticas de *QoS* (usando el comando **policy-map**).
3. Añadir la *service policy* a la interfaz con el comando **service-policy.**

3.4.2 Cómo definir una clase de tráfico (Traffic Class)

Como se ha visto, el primero de los pasos que hay que seguir para implementar los Servicios Diferenciados en el *router* usando el *MQC*, es definir una clase de tráfico para poder separar los diferentes flujos de tráfico cuando éstos alcancen el *router*. Para poder definir una clase de tráfico se empleará el comando **class-map**. Mediante las clases de tráfico, se podrán separar los paquetes que lleguen al *router* para aplicarles un tratamiento diferenciado. En este apartado se verá primero la sintaxis del comando y luego algunos ejemplos de configuración.

3.4.2.1.1 Configuración

La sintaxis del comando **class-map** es:

- **class-map [match-any|match-all]** *class-name*

- **no class-map [match-any|match-all]** *class-name*

El comando class-map:

- El comando **class-map** se usa para definir una clase de tráfico. Una clase de tráfico contiene principalmente tres elementos: un nombre, una serie de comandos **match** y una instrucción de cómo evaluar esos comandos **match**.

- El nombre se le da dentro de la línea del comando **class-map**. Por ejemplo, si se introduce el comando **class-map trafico_telnet** mientras se configura la clase de tráfico en el *CLI*, la clase se llamará trafico_telnet.

- El comando **class-map match-all** se usa cuando deben coincidir todos los criterios de selección para que un paquete entre a formar parte de la clase. El comando **class-map match-any** cuando sólo se debe cumplir uno de los criterios para que el paquete pertenezca a la clase.

El comando match:

- Los comandos **match** se usan para especificar los criterios para la clasificación de los paquetes. Los paquetes se comprueban para ver si cumplen con los criterios de selección especificados por los comandos **match**; si un paquete cumple el criterio especificado, el paquete será considerado miembro de la clase y será encaminado de acuerdo a las especificaciones de *QoS* que aparecen en la *service policy*. Los paquetes que no cumplen alguno de los criterios de selección son clasificados como miembros de la clase por defecto. La clase por defecto se explicará más adelante.

- Dentro de la clase especificaremos los criterios de selección de los paquetes usando el comando **match** seguido de:
  - **access-group** *access-group,* el criterio de selección se hará basándose en el número de la access-control list (*ACL*) especificado.
  - **mach any,** con este comando todos los paquetes pasarán a formar parte de la clase.
  - **mach class-map** *class-map-name,* para usar una clase como política de selección. El *Modular QoS CLI* permite a múltiples clases de tráfico (clases de tráfico anidadas, que son denominadas también class-maps anidadas), estar configuradas como una única clase.
  - **mach cos** *cos-value [cos-value cos-value cos-value],* para seleccionar paquetes basándose en el marcado de Clase de Servicio de la capa de enlace de datos.
  - **mach destination-address** *dirección-MAC,* usará la dirección MAC destino como criterio de selección.
  - **match input-interface** *interface-name,* utilizará la interfaz de entrada del paquete como criterio de selección.
  - **match ip dscp** *ip-dscp-value[ip-dscp-value ip-dscp-value ip-dscp-value ip-dscp-value ip-dscp-value ip-dscp-value ip-dscp-value],* identificará un determinado *IP DSCP* como criterio de selección
  - **match ip precedence** *ip-precedence-value[ip-precedence-value ip-precedence-value ip-precedence-value], í*dem que el anterior pero basándose en el valor del *IP* Precedence del paquete.
  - **match ip rtp** *starting-port-number port-range,* usará el Puerto del protocolo en tiempo real (RTP) como criterio de selección.
  - **match mpls experimental** *number,* para usar el valor del MPLS de los paquetes como criterio de selección.
  - **match not,** se usa para prevenir que un paquete pase a formar parte de una determinada clase.

- **match protocol** *protocol,* para configurar el criterio de selección de una clase basándose en el protocolo del paquete.
- **match qos-group** *qos-group-value,* para identificar un valor especifico de *QoS* como criterio de selección. El valor del grupo de *QoS* es local al *router*.
- **match source-address mac** *adress,* usará la dirección MAC origen como criterio de selección.

3.4.2.1.2 Ejemplos

***Access list como criterio de selección**:*

Creamos dos clases: Una clase llamada class1 en la que se clasificaran como pertenecientes a la clase aquellos paquetes que cumplan con la *ACL* 101 y una clase llamada class2 cuyos paquetes serán los que pasen por el filtro impuesto por la *ACL* 102.

```
Router(config)# class-map class1
Router(config-cmap)# match access-group 101
Router(config-cmap)# exit

Router(config)# class-map class2
Router(config-cmap)# match access-group 102
Router(config-cmap)# exit
```

Comandos match-any y match-all:

Ejemplo de configuración de los comandos **class-map match-any** y **class-map match-all.** Con el objetivo de mostrar la diferencia entre ambos comandos. Estos comandos determinan cómo serán evaluados los paquetes cuando existan varios criterios de selección por clase.

Configuración con el comando **class-map match-all**:

Si un paquete llega a un *router* y la clase de tráfico cisco1 está configurada en la interfaz, el paquete será evaluado para comprobar si cumple los 3 criterios de selección: pertenecer al protocolo *IP*, al grupo de *QoS* 4 y satisfacer las condiciones de la *ACL* 101. Todas las condiciones se deben cumplir para que el paquete entre a formar parte de la clase cisco1.

```
Router(config)# class-map match-all cisco1
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# match access-group 101
```

Configuración con el comando **class-map match-any**:

Si un paquete llega a un *router* y la clase de tráfico cisco2 está configurada en la interfaz, el paquete será evaluado para comprobar si cumple **alguno** de los 3 criterios de selección: pertenecer al protocolo *IP*, al grupo de *QoS* 4 y satisfacer las condiciones de la *ACL* 101. Si el paquete cumple alguno de los criterios, el paquete pasará a formar parte de la clase cisco2.

```
Router(config)# class-map match-any cisco2
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# match access-group 101
```

***Clases Anidadas; Una clase de tráfico como criterio de selección**:*

Hay dos razones para usar el comando **match class-map**. Una razón es el mantenimiento; si existe una clase de tráfico con una configuración muy extensa, es más fácil usar esa clase de tráfico como criterio de selección que volver a escribir de nuevo la misma configuración de tráfico.

Pero la razón más importante es permitir a los usuarios usar las características **match-any** y **match-all** en la misma clase de tráfico. Si se quiere hacer esta combinación, hay que crearse una clase de tráfico en la que se tenga activado match-any o match-all y después crear otra clase de tráfico que contenga a la anterior que tenga un criterio de selección diferente.

- **Ejemplo de Clases Anidadas para Mantenimiento.**

  En el siguiente ejemplo, la clase llamada class1 tiene las mismas características que la clase llamada class2, con la excepción que a la clase class1 se le ha añadido una dirección de destino como criterio de selección. Para no tener que repetir otra vez todos los criterios de selección de la clase2 en la clase1, simplemente usamos el comando **match class-map** dentro de la clase1.

```
Router(config)# class-map match-any class2
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 3
Router(config-cmap)# match access-group 2
Router(config-cmap)# exit
Router(config)# class-map match-all class1
Router(config-cmap)# match class-map class2
Router(config-cmap)# match destination-address mac 1.1.1
Router(config-cmap)# exit
```

- **Ejemplo de Clases Anidadas para combinar los comandos match-all y match-any:**

  Para que un paquete pase a formar parte de la clase class4 deberá cumplir: pertenecer al protocolo *IP* ‘y’ tener un valor 4 de grupo de *QoS* ‘o’ tener como dirección de destino MAC 1.1.1 ‘o’pertenecer al access group 2.

```
Router(config)# class-map match-all class3
Router(config-cmap)# match protocol ip
Router(config-cmap)# match qos-group 4
Router(config-cmap)# exit
Router(config)# class-map match-any class4
Router(config-cmap)# match class-map class3
Router(config-cmap)# match destination-address mac 1.1.1
Router(config-cmap)# match access-group 2
Router(config-cmap)# exit
```

Ejemplo del uso del comando match not:

El comando **match not** se utiliza para especificar que un valor determinado no será usado como criterio de selección. En el ejemplo pertenecerán a la clase todos aquellos paquetes cuyo protocolo no sea el protocolo *IP*.

```
Router(config)# class-map noip
Router(config-cmap)# match not protocol ip
Router(config-cmap)# exit
```

3.4.3 Como crear una Política *(Service Policy)*

El siguiente paso después de crear las clases en el *MQC* es crear las políticas, que especificarán el tratamiento que reciben los paquetes de cada una de las clases creadas. Este tratamiento pueden ser funciones policía, encolamiento, espaciado, marcado o cualquier otra función de *DiffServ*. El comando empleado para crear la política es **policy-map**. Mediante estas políticas se podrán aplicar las diferentes herramientas de *DiffServ* de Cisco como son: *CBWFQ*, *WRED*, Class-Based Paquet Marking, *Class-Based Policing*, etc, a los paquetes de cada una de las clases creadas con los comandos del apartado anterior.

3.4.3.1.1 Configuración

La sintaxis del comando **policy-map** es:

- **policy-map** *policy-name*
- **no policy-map** *policy-name*

La sintaxis del comando **class,** que usaremos cuando estemos dentro de la configuración de la *service policy* (después de poner el comando **policy-map),** es:

- **class** *class-name*
- **no class** *class-name*

El comando **policy-map** se utiliza asociado a una clase de tráfico que ha sido definida previamente con el comando **class-map**. El resultado de la asociación se denomina política o *service policy*. Una *service policy* tiene tres elementos principales:

- un nombre,
- una clase de trafico (especificada con el comando **class**),
- y las políticas de *QoS*.

El propósito de la *service policy* es asociar una clase de tráfico con una o más políticas de *QoS*.

Por tanto, los pasos a seguir a la hora de crear una *service police* son:

1. Crear la política con el comando **policy-map**
2. Especificar la clase sobre la que se aplicará la política con el comando **class**
3. Especificar el tratamiento que se le aplicaran a los paquetes de esa clase (política de *QoS*) *CBWFQ*, *WRED*, *LLQ*, *Class-Based Packet Marking*, etc.

El *Modular QoS CLI* no requiere necesariamente que los usuarios asocien una única clase de tráfico a una *service policy*. Cuando los paquetes coinciden para más de un criterio de selección, se pueden asociar múltiples clases de tráfico con una única *service policy*, es decir, el paso dos de la creación de la "*service police"* se puede repetir tantas veces como clases tengamos.

Si hay una clase por defecto configurada, todo el tráfico que no pertenece a ninguna de las otras clases se considerará perteneciente a esa clase por defecto. Esto es, que todo el tráfico que no cumpla los criterios de selección de alguna de las clases, se tratará dependiendo de la configuración de la política para la clase por defecto.

3.4.3.1.2 Ejemplos

Creación de una service police*:*

Se crea una *service policy* llamada policy1 en la que se tendrán dos clases. A los paquetes pertenecientes a la clase 1 se les dará un ancho de banda asegurado en caso de congestión de 3000 kbps y un tamaño de cola *FIFO* de 30 paquetes. Para los paquetes de la clase dos el ancho de banda asegurado en caso de congestión será de 2000 kbps.

```
Router(config)# policy-map policy1
Router(config-pmap)# class class1
Router(config-pmap-c)# bandwidth 3000
Router(config-pmap-c)# queue-limit 30
Router(config-pmap)# exit
Router(config-pmap)# class class2
Router(config-pmap-c)# bandwidth 2000
Router(config-pmap)# exit
```

Ejemplo de configuración de políticas de tráfico jerárquicas:

Una política de tráfico se puede anidar dentro de una política de *QoS* cuando el comando **service-policy** se utiliza dentro del modo de configuración de la política. Una política de tráfico que contiene una política de tráfico anidada se denomina una política de tráfico jerárquica.

La política hija es la que se define antes. La política padre será que contenga a la anterior.

En el siguiente ejemplo, la política hija es responsable de dar prioridad al tráfico y la política padre se encarga de espaciarlo (*shaping*).

```
Router(config)# policy-map hija
Router(config-pmap)# class voz
Router(config-pmap-c)# priority 50
Router(config)# policy-map padre
Router(config-pmap)# class class-default
Router(config-pmap-c)# shape average 10000000
Router(config-pmap-c)# service-policy hija
```

Ejemplo de configuración de la clase por defecto:

El tráfico sin clasificar, es decir, el tráfico que no se corresponde con ninguno de los criterios de selección de las clases de tráfico configuradas, se introduce en la clase por defecto.

Aunque el usuario no configure la clase por defecto los paquetes sin clasificar serán tratados como pertenecientes a la clase por defecto. Sin embargo, por defecto, la clase por defecto no tiene activadas ningún tipo de herramientas. Por lo tanto, los paquetes no recibirán *QoS*. Estos paquetes se colocarán en una cola *FIFO* y serán enviados a la tasa determinada por el ancho de banda disponible del enlace. Esta cola *FIFO* se controla mediante el mecanismo *Tail Drop*. *Tail Drop* es una forma de evitar la congestión que trata a todo el tráfico de la misma manera y no diferencia entre clases de servicio. Las colas se llenan durante periodos de congestión. Cuando la cola de salida está llena, los paquetes se descartan hasta que desaparece la congestión.

El siguiente ejemplo configura una política de tráfico para la clase por defecto de una política de tráfico llamada policy1.

La clase por defecto tiene las siguientes características: 10 colas para el tráfico sin clasificar, un máximo de 20 paquetes por cola antes de que *Tail Drop* comience a descartarlos.

```
Router(config)# policy-map policy1
Router(config-pmap)# class class-default
```

```
Router(config-pmap-c)# fair-queue 10
Router(config-pmap-c)# queue-limit 20
```

3.4.4 Como asociar una política (service policy) a una interfaz

El último paso a la hora de implementar los Servicos Diferenciados mediante el uso del *MQC*, es asociar la política creada en el paso anterior con una interfaz. Esto hará que esa política se aplique a los paquetes que entran o salén de una interfaz determinada. El comando para asociar la política a una interfaz es **service-policy**.

3.4.4.1.1 Configuración

La sintaxis del comando **service-policy** es:

- **service-policy [input|output]** *policy-map-name*
- **no service-policy [input|output]** *policy-map-name*

Usaremos el comando **service-policy,** dentro de la línea de comandos de configuración de la interfaz en cuestión para asociar la política, especificada previamente con el comando **policy-map,** con la interfaz determinada. Especificaremos también si se aplicará la política al tráfico entrante o al saliente usando las opciones **input** o **output**.

Usaremos la forma **no** del comando para desasociar la política de la interfaz.

3.4.4.1.2 Ejemplos

Ejemplo de como se asocia una política a una interfaz

Entramos dentro de la configuración de la interfaz y le asociamos la política policy1 para el tráfico saliente.

```
Router(config)# interface e1/1
Router(config-if)# service-policy output policy1
Router(config-if)# exit
```

3.4.5 Comandos para Verificar la Configuración

Una vez que se han configurado las políticas a aplicar a los paquetes de una determinada clase, para comprobar que el proceso se ha realizado correctamente se utilizarán los comandos:

- **Show class-map** *class-name,* para mostrar la información relativa a la clase de tráfico.
- **Show policy-map,** para mostrar la configuración de una política y sus clases de tráfico asociadas.

La siguiente tabla muestra una explicación más detallada de los comandos anteriores:

Tabla 3. 8: Comandos para la Verificación de las Configuraciones

Comando	Propósito
`Router# show class-map`	Muestra información de todas las clases de tráfico configuradas en el *router*.
`Router# show class-map class-name`	Muestra información de la clase de tráfico especificada
`Router# show policy-map`	Muestra todas las políticas de tráfico configuradas.
`Router# show policy-map policy-map-`	Muestra la política de tráfico, definida por usuario,

Comando	Propósito
name	especificada
Router# **show policy-map** interface	Muestra configuraciones y estadísticas de las políticas de entrada y salida asociadas a una interfaz
Router# **show policy-map** interface interface-spec	Muestra configuraciones y estadísticas de las políticas de entrada y salida asociadas a una interfaz particular
Router# **show policy-map** interface interface-spec input	Muestra configuraciones y estadísticas de la política de entrada asociadas a una interfaz
Router# **show policy-map** interface interface-spec output	Muestra configuraciones y estadísticas de la política de salida asociadas a una interfaz
Router# **show policy-map** [interface [interface-spec [input|output] [class class-name]]]]	Muestra la configuración y las estadísticas de la clase especificada configurada en la política.

3.4.6 Herramientas de Cisco para *DiffServ* configuradas usando el *MQC*

Cómo se expuso en el apartado 3.4, el *MQC* permite de forma sencilla, mediante el uso de una interfaz de comandos, configurar las distintas herramientas para implementar los *DiffServ* disponibles en el software Cisco *IOS*. Si se recuerda, del apartado 3.4.3 en el que se explicaba cómo configurar las políticas para las clases de tráfico, era dentro de los policy-maps donde se podían activar estas herramientas para aplicarlas a las diferentes clases de tráfico. En este capítulo se describen en detalle cada una de esas herramientas con sus comandos de configuración correspondientes.

3.4.7 *Class-Based Packet Marking*

Class-Based Packet Marking [27] proporciona un modo fácil mediante *CLI* para un marcado eficiente de los paquetes. Marca los paquetes de una determinada clase definida por el usuario previamente mediante el *MQC*. Se puede activar cuando se configura la política para una clase.

Ésta herramienta permite a los usuarios ejecutar las siguientes acciones;

- Marcar los paquetes poniendo los bits del *IP Precedence* o el *DSCP* en el *byte ToS* de la cabecera *IP*.
- Marcar los paquetes poniendo el valor de Clase de Servicio (*CoS*) de la Capa 2.
- Asociar un valor de grupo local de calidad de servicio a un paquete.
- Poner los bits de la prioridad de pérdida de celda (*Cell Loss Priority CLP*) en la cabecera *ATM* de un paquete de 0 a 1.
- Poner el bit de *Frame Relay Discard Eligibility* (*DE*) en el campo dirección del marco *frame relay* de 0 a 1.

En la mayoría de los casos el propósito del marcado de los paquetes es la identificación. Después de que un paquete se marque, los dispositivos del flujo de bajada identifican el tráfico basándose en ese marcado y lo tratan de acuerdo a las necesidades de la red. Esto ocurre cuando los comandos **match** dentro de las clases de tráfico están configurados para identificar los paquetes marcados. Por ejemplo, **match ip precedence, match ip dscp, math cos,** etc..

El marcado de paquetes te permite dividir la red en múltiples niveles o clases de servicio. *Class-Based Packet Marking* se usa frecuentemente para poner los valores de *IP Precedence* o el *IP DSCP* a los paquetes que entran en una red. Los elementos de red pueden usar esos valores marcados en los paquetes para determinar el tratamiento que se le aplicará al tráfico. Por ejemplo, el trafico de voz puede ser marcado con un valor determinado (*IP Precedence* o *DSCP*) y la herramienta de encolamiento de baja latencia *(Low Latency Queueing LLQ)* puede estar configurada para poner los paquetes marcados en una cola prioritaria *(priority queue)*.

También usaremos *Class-Based Packet Marking* para asignar paquetes a un grupo de *QoS* dentro del *router*. El valor del grupo de *QoS* normalmente es usado para una de las siguientes razones:

- Para aumentar el número de clases de tráfico. El grupo de *QoS* tiene 100 marcados diferentes de paquetes individuales, frente a *IP DSCP* e *IP Precedence* con 64 y 8 respectivamente.

- Si **no** queremos cambiar el valor del *IP Precedence* o del *IP DSCP* del paquete.

Esta herramienta de Cisco también permite hacer un mapeo de la Capa 2 a la Capa 3. Por ejemplo: si un paquete, que necesita ser marcado a un servicio de *QoS* diferenciado, está abandonando el *router* y entra en un *switch*, el *router* puede poner el valor de *CoS* del paquete ya que el *switch* puede procesar el marcado de *CoS* de la cabecera en la Capa 2 y viceversa.

Algo muy importante a tener en cuenta es que el *Class-Based Packet Marking* puede marcar sólo paquetes que circulan a través de caminos de conmutación de la herramienta de encaminamiento urgente de Cisco *(Cisco Express Forwarding CEF)*. Por lo tanto, para que funcione, *CEF* deberá estar configurado tanto en las interfaces que envían como en las que reciben paquetes.

Class-Based Packet Marking se puede configurar sobre una interfaz, subinterfaz, o un circuito virtual permanente *(PVC) ATM.* Pero no se soporta en las siguientes interfaces:

- *Fast EtherChannel*

- *Tunnel*

- *Primary Rate Interface (PRI) de la Red Digital de Servicios Integrados (RDSI)*

- *ATM switched virtual circuit (SVC)*

- *Frame Relay data-link connection identifier (DLCI)*

- Alguna interfaz que no soporte *CEF*.

También deberemos tener en cuenta que una política que contiene el comando **set qos-group** se puede asociar únicamente como política de entrada. No se emplea para paquetes que abandonan el *router*. Del mismo modo, una política que contiene el comando **set cos** se puede asociar sólo como política de salida.

3.4.7.1.1 Configuración

Los pasos para la configuración de la herramienta de Cisco *Class-Based Packet Marking*:

Tabla 3. 9: Comandos de Configuración de *Class-Based Packet Marking*

	Comando	Propósito
Paso 1	`Router(config)# policy-map policy-map`	Especifica el nombre de la política que va a ser creada.
Paso 2	`Router(config-pmap)# class class-map-name`	Especifica el nombre de la clase sobre la que se aplica la política.
Paso 3	`Router(config-pmap-c)# set [depende de como lo queramos marcar]`	Activa la herramienta *Class-Based Packet Marking* para que se marquen los paquetes de la clase. El argumento depende de lo que queramos marcar.
Paso 6	`Router(config)# interface nombre-de-la-interfaz`	Especifica el nombre de la interfaz a la que se asociará la política.
Paso 7	`Router(config-if)# service-policy input|output policy-map`	Asignamos la política creada a la interfaz.

Usaremos la negación **no set** *[depende de como lo queramos marcar]* para desactivar el marcado de paquetes de la clase.

En el paso 3 tenemos varias opciones:

- **set ip precedence**

  Para poner el valor del *IP Precedence* en la cabecera *IP* usaremos el comando **set ip Precedence** segudo de un numero entre 0 y 7 que pone el valor del *IP Precedence* en la cabecera *IP*. Para dejar el valor del *IP Precedence* como está usar la forma negativa del comando anterior.

  - **set ip precedence** *valor-de-ip-precedence*
  - **no set ip precedence**

  Una vez que los valores de *IP Precedence* están puestos, otros servicios de *QoS* como *WFQ* o *WRED* pueden operar con ellos.

- **set ip dscp**

  Para marcar un paquete poniendo el *IP DSCP* en el *byte ToS*, usaremos el comando **set ip dscp** más un número del 0 al 63 que especifica el valor del *IP DSCP*. Se pueden usar palabras reservadas como EF (*Expedited Forwarding*) y AF11 (clase AF11 de *Assured Forwarding*), en vez de valores numéricos. Para borrar un valor anterior de *IP DSCP* emplearemos la forma **no** del comando anterior.

  - **set ip dscp** *valor-de-ip-dscp*
  - **no set ip dscp** *valor-de-ip-dscp*

  Una vez que los bits *IP DSCP* están puestos en un paquete, otros servicios de *QoS* pueden operar con ellos.

  No se pueden marcar los paquetes con un valor de *IP Precedence* con el comando **set ip precedence** y después marcarlos con un valor de *IP DSCP* con el comando **set ip dscp.**

- **set qos-group**

Para poner un identificador de grupo de *QoS (group ID),* que se puede usar después para clasificar paquetes, utilizaremos el comando **set qos-group**. Para borrar el *group ID* utilizaremos la forma **no** del comando anterior.

- **set qos-group** *group-id*
- **no set qos-group** *group-id*

  group-id -> Número del Group ID en el rango de 0 a 99.

Este comando permite asociar un grupo de *QoS* con un paquete. El identificador del grupo se puede usar después para clasificar paquetes en grupos de *QoS*.

- **set cos** *cos-value* especifica el valor de *CoS* asociado al paquete. El número está en el rango de 0 a 7

3.4.7.1.2 Ejemplos

Marcar los paquetes con el valor de IP Precedence

El siguiente ejemplo pone el valor de *IP Precedence* a 5 de los paquetes que satisfagan los criterios de selección de la clase class1:

```
Router(config)# policy-map policy1
Router(config-pmap)# class class1
Router(config-pmap-c)# set ip precedence 5
```

Todos los paquetes que satisfagan los criterios de selección de la clase 1 se marcarán con el valor de *IP Precedence* 5. El tratamiento que reciban los paquetes dependerá de la configuración de la red. Nótese que para marcar los paquetes de una determinada clase, primero se debe de haber creado la clase con el comando **class-map** y después la política para esa clase con el comando **policy-map**.

Marcar los paquetes con el valor de IP DSCP

En el siguiente ejemplo le daremos el valor 8 al *IP DSCP* en la política denominada policy1:

```
Router(config): policy-map policy1
Router(config-pmap)# class class1
Router(config-pmap-c)# set ip dscp 8
```

Todos los paquetes que satisfagan los criterios de selección de la clase 1 se marcarán con el valor de *IP DSCP* de 8. El tratamiento de estos paquetes a lo largo de la red dependerá de la red. Nótese que para marcar los paquetes de una determinada clase, primero se debe de haber creado la clase con el comando **class-map** y después la política para esa clase con el comando **policy-map**.

Marcar los paquetes con el valor del grupo de QoS

El siguiente ejemplo pone el valor del grupo de *QoS* a 1 para todos los paquetes que pertenecen a la clase 1.

```
Router(config): policy-map policy1
```

```
Router(config-pmap)# class class1
Router(config-pmap-c)# set qos-group 1
```

Nótese que para marcar los paquetes de una determinada clase, primero se debe de haber creado la clase con el comando **class-map** y después la política para esa clase con el comando **policy-map**.

3.4.8 Traffic *Policing*

Traffic Policing [16] permite limitar la tasa de transmisión de una clase de tráfico. Para ello se basa en criterios definidos por el usuario. También permite al sistema marcar los paquetes con el *IP DSCP* o el *IP Precedence*. Por ejemplo: Cuando un paquete sobrepasa la tasa contratada hay que marcarlo con un *DSCP* diferente que en el caso de que el paquete cumpla la tasa contratada. Esto permitirá a la red deshacerse de este tipo de paquetes en caso de congestión. Notar que en el caso anterior (*Class-Based Packet Marking)* los paquetes se marcaban si pertenecían a una clase determinada definida por el usuario.

Traffic Policing permite controlar la tasa máxima transmitida o recibida sobre una interfaz. De este modo se podrá controlar el ancho de banda del enlace. *Traffic Policing* se configura frecuentemente sobre interfaces en los extremos de la red para limitar el tráfico que entra o sale de ella. En la mayoría de las configuraciones de *Traffic Policing*, el tráfico que cae dentro de los parámetros acordados es transmitido, mientras que el que excede es descartado o transmitido con una prioridad diferente.

Traffic Policing puede realizar un marcado o remarcado de los paquetes. Como veíamos en la anterior herramienta, el *Class-Based Packet Marking,* el marcado de paquetes permite dividir la red en múltiples niveles de prioridad o clases de servicio:

- Podemos usar *Traffic Policing* para poner los valores del *IP DSCP* o del *IP Precedence* a los paquetes que entran a la red.

- También para asignar los paquetes a un grupo *QoS*. El *router* usara estos grupos *QoS* par dar prioridad a los paquetes dentro del mismo *router*.

De nuevo, *Traffic Policing* puede monitorizar sólo caminos de conmutación del *Cisco Express Forwarding (CEF)*. Por lo tanto, para que funcione, *CEF* deberá estar configurado tanto en las interfaces que envían como en las que reciben paquetes.

Traffic Policing se puede configurar sobre una interfaz, subinterfaz, o un circuito virtual permanente *(PVC) ATM*. Pero las siguientes interfaces no soportan el *Traffic Policing*:

- *Fast EtherChannel*

- *Tunnel*

- *Primary Rate Interface (PRI) de la Red Digital de Servicios Integrados (RDSI).*

- Alguna interfaz que no soporte *CEF*.

- Sobre la serie de Cisco *router* 7500, *Traffic Policing* no puede aplicarse a paquetes que se originan desde o van destinados a un *router*.

Cisco *IOS* soporta los siguientes métodos de *Traffic Policing* [34]:

- *Commited Access Rate (CAR)* [35]
- *Class-Based Policing* [16]

Los dos mecanismos tienen diferencias importantes funcionalmente, como veremos en la comparación entre ambos. Cisco recomienda *Class-Based Policing* y otras herramientas del *MQC* cuando aplicamos políticas de *QoS*. Por esta razón, se explicará la configuración de esta herramienta. Una comparación entre ambas herramientas la podremos encontrar en el anexo B de este documento.

3.4.8.1.1 Configuración de *Class-Based Policing*

Para configurar satisfactoriamente *Class-Based Policing* [16], se deberán tener creadas una clase y una política, y la política debe estar asociada a una interfaz especifica. Todo esto se realiza, como ya se vio, usando el *Modular QoS CLI*.

Class-Based Policing trabaja con un mecanismo de *token bucket*. Hay dos tipos de algoritmos de *token bucket*: un algoritmo de un solo *token bucket* y un algoritmo de doble *token bucket*. Cisco puede implementar estos dos algoritmos. Cisco *IOS* implementa un sistema de un *token bucket* cuando la opción **violate-action** no se especifica, y un sistema de doble *token bucket* cuando la opción **violate-action** si se especifica.

Para configurar *Class-Based Policing*, se usará el comando **police**. Se empleará la forma **no** del comando **police** para desactivar *Class-Based Policing* de la política de la clase.

Los pasos a seguir para configurar *Class-Based Policing* son:

Tabla 3. 10: Comandos de Configuración de *Class-Based Policing*

	Comando	Propósito
Paso 1	`Router(config)# policy-map policy-map`	Especifica el nombre de la política que va a ser creada.
Paso 2	`Router(config-pmap)# class class-map-name`	Especifica el nombre de la clase sobre la que se aplica la política.
Paso 3	`Router(config-pmap-c)# police bps burst-normal burst-max conform-action action exceed-action action [violate-action action]`	Activa la herramienta *Class-Based Policing* para que se aplique el algoritmo de *token bucket* sobre los paquetes de la clase.
Paso 6	`Router(config)# interface nombre-de-la-interfaz`	Especifica el nombre de la interfaz a la que se asociará la política.
Paso 7	`Router(config-if)# service-policy input|output policy-map`	Asignamos la política creada a la interfaz.

Los argumentos del comando **police** son:

bps (Requerido), tasa media en bits por segundo
burst-normal (Requerido), tamaño de ráfaga normal en *bytes* (tamaño del cubo)

burst-max (Opcional), *excess burst size* en *bytes*. En Cisco *IOS*, el *excess burst size* no tiene que estar especificado a menos que la opción **violate-action** esté también especificada.

Conform-action (Requerido), acción sobre los paquetes que estén conformes con la tasa límite.
Exeed-action (Requerido), acción llevada a cabo sobre los paquetes que excedan la tasa límite.
Violate-action (Opcional), acción sobre los paquetes que violan el tamaño de ráfaga normal y máximo. Si la opción **violate-action** esta especificada, el algoritmo *token bucket* trabaja con un algoritmo de doble *token bucket*.

Action, acción a llevar a cabo con los paquetes. Especificada por uno de las siguientes palabras:

- **drop,** tira el paquete.
- **set-clp-transmit,** pone el bit *ATM Cell Loss Priority (CLP)* de 0 a 1 sobre la celda *ATM* y transmite el paquete con el bit *ATM CLP* puesto a 1.
- **set-dscp-transmit,** pone el valor *IP DSCP* y transmite el paquete con el nuevo valor de *IP DSCP*.
- **set-frde-transmit**, pone el bit *Frame Relay Discard Eligibility (DE)* de 0 a 1 en el marco *frame relay* y transmite el paquete con el bit *DE* puesto a 1.
- **set-mpls-exp-transmit,** pone los bits experimentales *MPLS* (0 a 7) y transmite el paquete con el nuevo valor de bit experimental *MPLS* puesto.
- **set-prec-transmit** *new-prec,* pone el valor del *IP Precedence* y transmite el paquete con el nuevo valor de *IP Precedence* puesto.
- **set-qos-transmit** *new-qos,* pone el valor del grupo *QoS* y transmite el paquete con el nuevo valor de grupo de *QoS* puesto.
- **transmit,** transmite el paquete. El paquete se transmite sin ser alterado.

3.4.8.1.2 Funcionamiento de los algoritmos de Token Bucket en el Cisco *IOS*.

Notar que el mecanismo de *token bucket* utilizado para *Traffic Shaping* tiene un *token bucket* y un buffer de datos o cola; si no tuviera el buffer de datos sería un *Traffic Policing*. Para *Traffic Shaping*, los paquetes que llegan que no pueden ser enviados inmediatamente son retardados en el buffer de datos (esta herramienta se verá más adelante).

3.4.8.1.2.1 Algoritmo de Token Bucket con un Token Bucket

Este algoritmo se usa cuando la opción **violate-action** no está especificada en el comando CLI **police.**

El tamaño del *conform bucket* se inicializa al tamaño máximo especificado por el número de *bytes* del tamaño normal de ráfaga.

Cuando un paquete de tamaño B *bytes* llega en el instante t, ocurre lo siguiente:

a. Los *tokens* o testigos son renovados en el *conform bucket*. Si la llegada anterior del paquete fue en el instante t1 y el instante actual es t, el cubo (*bucket*) es renovado con el valor de bits (t-t1) basándose en la tasa de llegada de *tokens*. La tasa de llegada de *tokens* se calcula así:

   Tiempo entre paquetes = t-t1

$$((t - t1) \cdot \text{tasa de generación de tokens})/8 \text{ bytes}$$

b. Se restan entonces el número de *bytes* del *conform bucket* menos el número de *bytes* del paquete. Si el resultado de la diferencia es mayor o igual a cero, se lleva a cabo la *conform action* para ese paquete y se borran los *bytes* correspondientes al tamaño del paquete del *conform bucket*.

c. Si el resultado de la diferencia anterior es menor que 0, se lleva a cabo la *exceed action*.

3.4.8.1.2.2 Algoritmo de Token Bucket con 2 Token Buckets

Éste algoritmo se usa cuando la opción **violate-action** está especificada en el comando *CLI* **police.**

El tamaño del *conform bucket* se inicializa al tamaño máximo especificado por el número de *bytes* del tamaño normal de ráfaga.

El tamaño del *exceed bucket* se inicializa al tamaño máximo especificado por el número de *bytes* del tamaño máximo de ráfaga.

Los *tokens* para ambos *token buckets* son renovados basándose en la tasa de llegada de *tokens*.

Cuando un paquete de tamaño B *bytes* llega en el instante t, ocurre lo siguiente:

a. Los *tokens* son renovados en el *conform bucket*. Si la llegada anterior del paquete fue en el instante t1 y el instante actual es t, el cubo (*bucket*) es renovado con el valor de bits (t-t1) basándose en la tasa de llegada de *tokens*. Antes de desbordar el *conform bucket* los *tokens* que llegan son colocados en él. Cuando el *conform bucket* está lleno, los *tokens* nuevos se colocan en el *exceed bucket*.

   La tasa de llegada de *tokens* se calcula así:

   Tiempo entre paquetes = t-t1

$$((t - t1) \cdot \text{tasa de generación de tokens})/8 \text{ bytes}$$

b. Se restan entonces el número de *bytes* del *conform bucket* menos el número de *bytes* del paquete. Si el resultado de la diferencia es mayor o igual a cero, se lleva a cabo la *conform action* para ese paquete y se borran los *bytes* correspondientes al tamaño del paquete del *conform bucket*. El *exceed bucket* no se ve afectado en este caso.

c. Si el resultado de la diferencia anterior es menor que 0, se restan entonces el numero de *bytes* del *exceed bucket* menos el numero de *bytes* del paquete. Si el resultado de la diferencia es mayor o igual a cero, se lleva a cabo la *exceed action* para ese paquete y se borran los *bytes* correspondientes al tamaño del paquete del *exceed bucket*. No se borran *bytes* del *conform bucket*.

d. Si el resultado de la diferencia anterior es menor que 0, el paquete viola y se ejecuta la *violate action* para ese paquete.

Los valores recomendados para el *normal burst* y el *exceed burst* son:

- $\text{normal burst} = \text{tasa configurada} * (1 \text{ byte})/(8 \text{ bits}) * 1{,}5 \text{ segundos}$
- $\text{extended burst} = 2 * \text{normal burst}$

3.4.8.1.3 Ejemplos

Algoritmo de token bucket con un token bucket

En este ejemplo en particular, *Traffic Policing* lo configuramos con una tasa media de 8000 bps y el tamaño normal de ráfaga es 1000 *bytes* para todos los paquetes que salen de la interfaz *Fast Ethernet 0/0*.

```
Router(config)# class-map access-match
Router(config-cmap)# match access-group 1
Router(config-cmap)# exit
Router(config)# policy-map police-setting
Router(config-pmap)# class access-match
Router(config-pmap-c)# police 8000 1000 conform-action transmit exceed-
action drop
Router(config-pmap-c)# exit
Router(config-pmap)# exit
Router(config)# interface fastethernet 0/0
Router(config-if)# service-policy output police-setting
```

Estos paquetes se tratan basándose en las reglas vistas en el Capítulo en el que se explicaban los algoritmos de *token bucket* con un sólo *token*.

En este ejemplo, el tamaño del *token bucket* comienza lleno (1000 *bytes*). Si llega un paquete de 450 *bytes*, el paquete está conforme ya que hay suficientes *bytes* en el *token bucket*. Se lleva a cabo la *conform action* (transmitir) para ese paquete y se borran 450 *bytes* del *token bucket* (quedando 550 *bytes*).

Si el próximo paquete llega 0.25 segundos más tarde, 250 *bytes* son añadidos al *token bucket* (0.25 * 8000)/8), quedando 800 *bytes* en el *token bucket*. Si el siguiente paquete tiene un tamaño de 900 *bytes*, el paquete excede y se lleva a cabo la *exceed action* (tirarlo). No se borran *bytes* del *token bucket*. La siguiente figura muestra de forma gráfica el ejemplo:

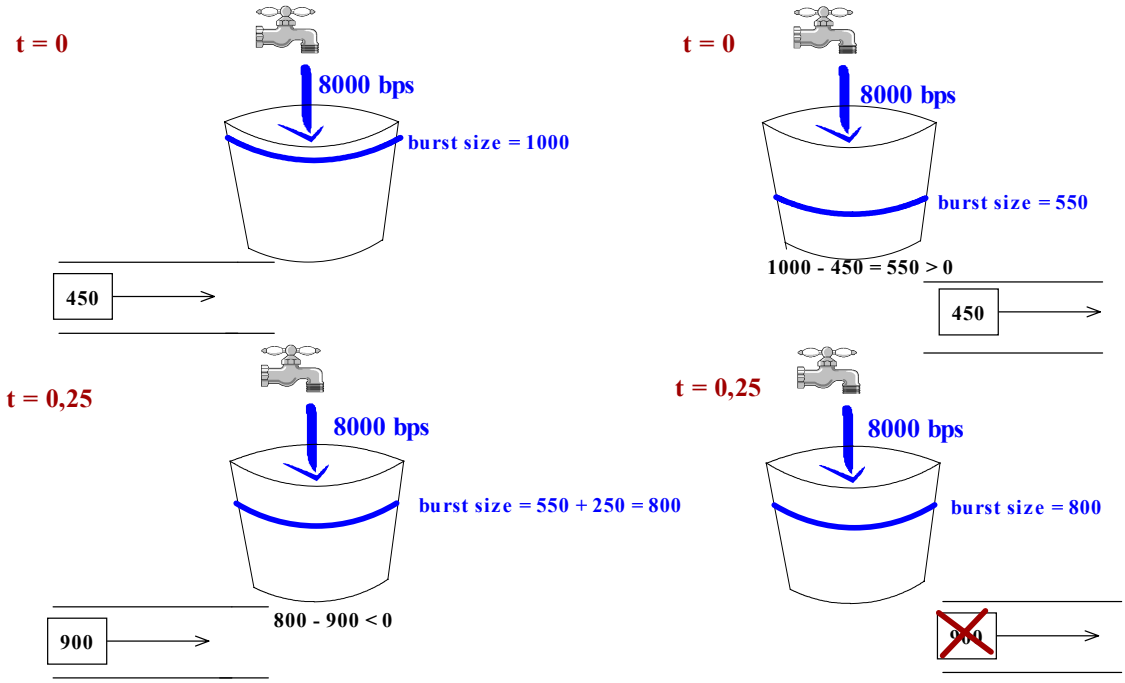

Figura 3. 7: Ejemplo de Algoritmo de Token Bucket con un único Token Bucket

Algoritmo de token bucket con doble token bucket

En este ejemplo en particular, *Traffic Policing* lo configuramos con una tasa media de 8000 bps, el tamaño normal de ráfaga es 1000 *bytes*, y el tamaño del *exceed burst* es de 1000 *bytes* para todos los paquetes que salen de la interfaz *Fast Ethernet 0/0*.

```
Router(config)# class-map access-match
Router(config-cmap)# match access-group 1
Router(config-cmap)# exit
Router(config)# policy-map police-setting
Router(config-pmap)# class access-match
Router(config-pmap-c)# police 8000 1000 1000 conform-action transmit
exceed-action set-qos-transmit 1 violate-action drop
Router(config-pmap-c)# exit
Router(config-pmap)# exit
Router(config)# interface fastethernet 0/0
Router(config-if)# service-policy output police-setting
```

Estos paquetes se tratan basándose en las reglas vistas anteriormente.

En este ejemplo, el tamaño del *token bucket* comienza lleno (1000 *bytes*). Si llega un paquete de 450 *bytes*, el paquete está conforme ya que hay suficientes *bytes* en el *token bucket*. Se lleva a cabo la *conform action* (transmitir) para ese paquete y se borran 450 *bytes* del *token bucket* (quedando 550 *bytes*).

Si el próximo paquete llega 0.25 segundos más tarde, son añadidos 250 *bytes* al *token bucket* (0.25 * 8000)/8), quedando 800 *bytes* en el *token bucket*. Si el siguiente paquete tiene un tamaño de 900 *bytes*, el paquete no está conforme ya que sólo hay 800 *bytes* disponibles en el *conform bucket*.

El *exceed bucket*, que comienza lleno con 1000 *bytes* se chequea para ver si tiene *bytes* disponibles. Debido a que hay suficientes *bytes* disponibles en el *exceed bucket*, se realiza la *exceed action* (poner el valor de *QoS* a 1) y 900 *bytes* se borran del *exceed bucket* (quedando 100 *bytes*).

Si el siguiente paquete llega 0.40 segundos después, 400 *bytes* se añaden al *token bucket* ((0.40*8000)/8). Por lo tanto, el *conform bucket* dispone ahora de 1000 *bytes* (el máximo) y los 200 *bytes* que sobran se pasan al *exceed bucket* quedando este con 300 *bytes*.

Si el paquete que llega es de 1000 *bytes*, el paquete está conforme ya que hay suficientes *bytes* libres en el *conform bucket*. Se lleva a cabo la *conform action* para ese paquete (transmitirlo) y se borran los 1000 *bytes* del *conform bucket*.

Si el siguiente paquete llega 0.20 segundos después, se añaden 200 *bytes* al *token bucket* ((0.20*8000)/8). Por lo tanto, el *conform bucket* tiene ahora 200 *bytes*. Si el paquete que llega es de 400 *bytes*, el paquete no está conforme ya que sólo hay 200 *bytes* disponibles en el *conform bucket*. De igual modo, el paquete tampoco excede debido a que sólo hay 300 bytes disponibles en el *exceed bucket*. Así, al paquete se le aplicará la *violate action* (tirarlo).

3.4.9 *Traffic Shaping*

Traffic Shaping [28] permite controlar el tráfico que abandona una interfaz para casar su flujo con la velocidad de la interfaz remota, y asegurar así que el tráfico cumpla las políticas contratadas para él. Esto permite eliminar los cuellos de botella en las topologías. *Traffic Shaping* dispone de un mecanismo de *token bucket* y de memorias para almacenar paquetes. Cuando llega una ráfaga de tráfico la almacena y la sirve a una tasa constante con lo que suaviza las crestas de tráfico producidas por estas ráfagas, espaciando los paquetes que le llegan en el tiempo.

Principales razones para usar *Traffic Shaping* son:

- Controlar el acceso al ancho de banda disponible.
- Asegurar que el tráfico cumple las políticas establecidas para él.
- Evitar la congestión que puede ocurrir cuando enviamos un exceso de tráfico a una interfaz remota.

Traffic Shaping previene la pérdida de paquetes. Su uso es especialmente importante en redes *Frame Relay* ya que el *switch* no puede determinar el orden de los paquetes y por lo tanto los paquetes son tirados cuando hay congestión. Más aún, es de vital importancia para tráfico en tiempo real tal como voz sobre *Frame Relay*. Retener los datos en el *router* le permite a éste priorizar el tráfico.

Traffic Shaping limita la tasa de transmisión de datos. Puedes limitar la tasa de datos a uno de los siguientes:

- Una tasa específica configurada.
- Una tasa derivada del nivel de congestión.

La tasa de transferencia depende de tres componentes que constituyen el *token bucket*: tamaño de ráfaga *(burst size)*, tasa media y el intervalo de medida (tiempo). La tasa media es igual al tamaño de ráfaga dividido por el intervalo.

Una variable adicional se le aplica al *Traffic Shaping*: *Be size* (el tamaño de ráfaga de exceso *(the exceed burst size)*). El *Be size* permite estar enviando más tráfico que el del tamaño normal de ráfaga durante un intervalo de tiempo en ciertas situaciones. Se permitirá el paso de los paquetes que provengan del *Excess Burst* pero serán marcados.

Traffic Shaping suaviza el tráfico almacenando tráfico a la tasa configurada en una cola.
Cuando un paquete llega a la interfaz para transmitirse, ocurre lo siguiente:

1. Si la cola está vacía, el paquete es procesado por el *Traffic Shaper*:
   - Si es posible, el *Traffic Shaper* envía el paquete.
   - En otro caso, el paquete se coloca en la cola.
2. Si la cola no está vacía, el paquete se coloca en la cola.

Cuando los paquetes están en la cola, el *Traffic Shaper* borra, el número de paquetes que puede enviar, de la cola cada intervalo de tiempo.

El anexo B proporciona información sobre las herramientas que Cisco ofrece para realizar *Traffic Shaping.*

3.4.9.1.1 *Policing* vs *Shaping* [36]

El siguiente diagrama muestra la diferencia clave: *Traffic Policing* propaga la ráfaga. Cuando la tasa de tráfico alcanza la tasa máxima configurada, el exceso de tráfico se tira (o se remarca). El resultado es una tasa de salida que parece un diente de sierra con crestas y depresiones. En contraste con el *policing*, *Traffic Shaping* retiene el exceso de paquetes en una cola o *leaky bucket* y entonces programa ese exceso para posteriores transmisiones a costa de incrementar el tiempo. El resultado del *Traffic Shaping* es una tasa de paquetes de salida suavizada.

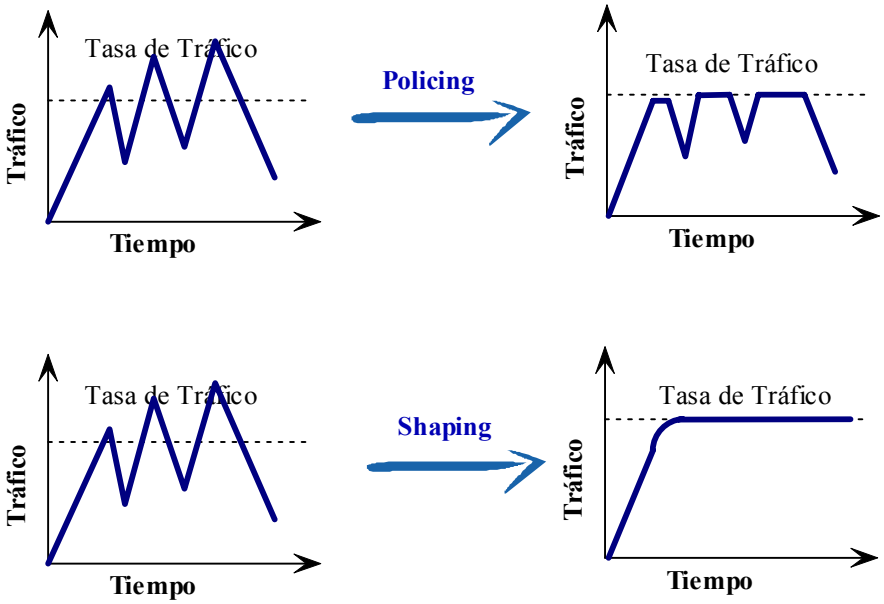

Figura 3. 8: Diferencias entre *Policing* y *Shaping*

Shaping implica la existencia de una cola con suficiente memoria para almacenar los paquetes retardados, mientras que *policing* no. Por lo tanto, hay que asegurarse de tener suficiente memoria disponible cuando se activa *shaping*. Además, *shaping* requiere una función de programación para posteriores transmisiones de alguno de los paquetes retardados. Por estos motivos, reflejados en la siguiente tabla, se ha optado en este trabajo por implementar funciones policía en vez de espaciadores en los routers. Considerando la posibilidad de realizar trabajos futuros que incluyan comparativas experimentales de ambos mecanismos.

Tabla 3. 11: Resumen de Diferencias entre *Policing* y *Shaping*

	Shaping	***Policing***
Objetivo	Almacenar y encolar los paquetes que excedan las tasas acordadas	Tirar (o remarcar) los paquetes que excedan las tasas acordadas. (no almacena)
Tasa de Refresco de Tokens	Incrementada al comienzo del intervalo de tiempo. (se necesita el número mínimo de intervalos).	Continuo basado en la fórmula: 1/tasa de información acordada
Valores de Los Tokens	Configurada en bits por segundo	Configurada en bytes.
Opciones de Configuración	·**shape** comando usado en el *MQC* para implementar *Class-*	·**police** comando usado en el *MQC* para implementar *traffic policing* (en clases).

	Shaping	*Policing*
	Based Shaping. ·**frame-relay traffic-shape** comando para implementar *FRTS* ·**traffic-shape** comando para implementar *GTS*.	·**rate-limit** comando para implementar committed access rate (CAR).
Aplicable sobre tráfico entrante.	No	Si
Aplicable sobre tráfico saliente	Si	Si
Ráfagas	Controla las ráfagas suavizando la tasa de salida sobre al menos ocho intervalos de tiempo. _Usa un *leaky bucket* para retardar el tráfico, logrando un efecto suavizante.	Propaga la ráfaga. No suaviza.
Ventajas	Encolar el exceso de tráfico en vez de tirarlo. Normalmente evita retransmisiones debidas al descarte de paquetes.	Controla la tasa de salida a través del descarte de paquetes. Evita retardos debidos al encolado.
Desventajas	Puede introducir retardos debidos al encolado, particularmente en colas grandes.	Tirar los excedentes de tráfico (cuando está configurado), estrangula el tamaño de la ventana *TCP* y reduce la tasa total de salida de los flujos afectados. Tamaños de ráfaga demasiado agresivos pueden conducir a tiradas excesivas de paquetes y estrangular la tasa total de salida, particularmente en los flujos basados en *TCP*.
Remarcado Opcional de Paquetes	No	Si

3.4.10 *Class-Based Weighted Fair Queueing (CBWFQ)*

Class-Based Weighted Fair Queueing [24][29][30] es un mecanismo usado para proporcionar un ancho de banda mínimo garantizado a las diferentes clases de tráfico durante periodos de congestión. *CBWFQ* extiende la funcionalidad estándar de *Weighted Fair Queueing (WFQ)* para dar soporte a clases de tráfico definidas por el usuario. Para *CBWFQ*, se definen clases de trafico basadas en criterios de selección incluyendo protocolos, listas de control de acceso *(access control lists ACLs)* (ver anexo A) e interfaces de entrada. Los paquetes que satisfagan los criterios de selección para una clase constituyen el tráfico para esa clase. Se reserva una cola para cada clase, y el tráfico perteneciente a una clase se mete directamente en la cola de esa clase.

Una vez que la clase ha sido definida de acuerdo con los criterios de selección, se le pueden asignar características. Para caracterizar una clase, se le asigna su ancho de banda, peso y límite máximo de paquete. El ancho de banda asignado para la clase es el ancho de banda garantizado para esa clase durante periodos de congestión.

Para caracterizar una clase también se puede especificar el tamaño de su cola, que es el número máximo de paquetes que se permiten acumular en la cola de dicha clase. Los paquetes pertenecientes a una clase están sujetos al ancho de banda y el tamaño de cola asignados para esa clase.

Después de que una cola alcance el tamaño máximo de cola configurado, el encolado de nuevos paquetes de la clase causa que se produzca *Tail Drop* o *packet drop* con *WRED*, dependiendo de

la política que esté configurada. Las clases que usan *CBWFQ* utilizan *Tail Drop* por defecto, a menos que se configure explícitamente *Weighted Random Early Detection (WRED)* para tirar los paquetes.

Un dato muy importante a tener en cuenta es que si se utiliza *WRED* dentro de la clase, no debe estar configurada *WRED* en la interfaz a la que se le aplica la *service policy*.

El estándar de *WFQ* es la clasificación por flujos. Esto es, los paquetes con la misma dirección *IP* origen, dirección *IP* destino, puerto origen *TCP* o *UDP* o puerto destino *TCP* o *UDP*, son clasificados como pertenecientes al mismo flujo. *WFQ* reserva una cantidad igual de ancho de banda para cada flujo. Al *Flow-based WFQ* se le denomina también de encolamiento justo ya que todos los flujos tienen el mismo peso.

Para *CBWFQ*, que extiende del estándar de *WFQ*, el peso especificado para la clase será el peso de cada paquete que cumpla los criterios de selección de la clase. Los paquetes que llegan a la interfaz de salida son clasificados de acuerdo con los filtros de selección que definimos, entonces a cada uno se le asigna el peso apropiado. El peso para un paquete que pertenece a una clase determinada se deriva del ancho de banda que se le asignó a la clase cuando se configuró; en ese sentido el peso de cada clase es configurable por el usuario.

Después de que se haya asignado el peso de un paquete, el paquete es encolado en la cola de la clase apropiada. *CBWFQ* utiliza los pesos asignados a los paquetes de la cola para asegurarse de que la cola de la clase es servida de manera justa.

CBWFQ permite especificar exactamente la cantidad de ancho de banda a reservar para una clase determinada de tráfico. Teniendo en cuenta el ancho de banda disponible de la interfaz., podemos configurar hasta 64 clases y controlar la cantidad de ancho de banda para cada una, no como en el caso del *flow-based WFQ*.

Flow-based WFQ aplica pesos al tráfico clasificado en conversaciones y determina el ancho de banda permitido a cada conversación en relación con otras conversaciones. Para *flow-based WFQ*, estos pesos y clasificaciones, son dependientes y están limitados a los siete niveles de *IP Precedence*.

CBWFQ permite definir qué constituye una clase, basándose en criterios que exceden los confines del flujo. *CBWFQ* permite utilizar *ACLs* y protocolos o nombres de interfaces de entrada para definir cómo será clasificado el tráfico. No se necesita mantener la clasificación del tráfico basada en flujo. Además, se pueden configurar hasta 64 clases discretas en una *service policy*.

Se debe tener en cuenta que configurar *CBWFQ* sobre una interfaz física es posible sólo si la interfaz tiene activado el encolamiento por defecto, es decir, para las interfaces serial E1 (2.048Mbps) e inferiores *WFQ* (otras interfaces usan *FIFO*). Activar *CBWFQ* en una interfaz física anula el método de encolado por defecto, mientras que activar *CBWFQ* sobre un *ATM PVC* no lo anula.

CBWFQ no soporta a la vez *Traffic Shaping* y *Policing*. En cuanto a las interfaces ATM, sólo soportan *CBWFQ* aquellas conexiones de tasa de bits variable (*variable bit rate VBR)* y de tasa de bit disponible *(available bit rate ABR)*. No estando disponible para conexiones *ATM* de tasa de bits no especificada *(unspecified bit rate UBR)*. Obsérvese también que en subinterfaces tampoco se soporta *CBWFQ*.

Finalmente, no hay que olvidar que para poder usar *CBWFQ* es necesario asegurarse de que *WFQ* no está activo para la interfaz, asegurarse de que *WRED* no está activo para la interfaz y conocer cómo se configuran las *ACLs*.

3.4.10.1.1 Configuración

Para configurar *CBWFQ* se deben realizar tres pasos fundamentalmente: definir las *class-maps* o clases de tráfico, configurar la política que activará *CBWFQ* en esas clases y asociar la política a una interfaz. La forma de definir las clases y las políticas ya se vio en el apartado 3.4.1. Una vez que las clases están definidas los pasos a seguir son los que se muestran en la siguiente tabla:

Tabla 3. 12: Pasos psra la Configuración de *CBWFQ*

	Comando	Propósito
Paso 1	`Router(config)# policy-map policy-map`	Especifica el nombre de la política que va a ser creada.
Paso 2	`Router(config-pmap)# class class-map-name`	Especifica el nombre de la clase sobre la que se aplica la política.
Paso 3	`Router(config-pmap-c)# bandwidth {bandwidth-kbps|percent percent}`	Activa *CBWFQ* en la clase, reservando una cantidad de ancho de banda para la clase especificada en kbps o en un porcentaje del total de la interfaz
Paso 5	`Router(config)# interface nombre-de-la-interfaz`	Especifica el nombre de la interfaz a la que se asociará la política.
Paso 6	`Router(config-if)# service-policy input|output policy-map`	Asignamos la política creada a la interfaz.

Una vez dentro del comando **policy-map** se accederá a la clase con el comando **class** *nombre-clase*. Ahora, se usará el comando **bandwidth** *bandwith-kbps* con él se especifica la cantidad de kbps que serán asignados a la clase, es decir, el peso que tendrán los paquetes que pertenezcan a la cola de esa clase.

Una vez configurado *CBWFQ* mediante el comando anterior, se puede configurar la clase para que use *Tail Drop* o *WRED* como modo de descartar paquetes. El comando para usar *Tail Drop* es **queue-limit** *number-of-packets* y el comando a usar si se quiere aplicar *WRED* como método para tirar paquetes es **random-detect** (aunque las opciones de *WRED* se verán más adelante)**.** Notar que por defecto se establecerá *Tail Drop* como método para tirar paquetes, aunque no se emplee el comando **queue-limit.**

También se puede configurar la política de la clase por defecto, que es aquella a la que los paquetes que no pertenecen a ninguna de las clases configuradas son enviados. Por regla general, la clase por defecto está configurada con *WFQ flow-based* como política de cola. Sin embargo, se podrá configurar con el comando **bandwidth** que anulará al *WFQ flow-based*. En resumen, se utilizará el comando **bandwidth** *bandwith-kbps* si queremos emplear *CBWFQ* o el comando **fair-queue** *[numero de colas dinámicas]* si lo que queremos configurar es *flow-based WFQ*.

En la clase por defecto también se podrá elegir si se quiere configurar *Tail Drop* o *WRED*.

Comando bandwidth:

Para especificar o modificar el valor del ancho de banda reservado para una clase utilizaremos el comando **bandwidth** junto con la cantidad de ancho de banda en kilobits por segundo asignados a la clase. Para eliminar el ancho de banda especificado para una interfaz usaremos la forma **no** de ese comando.

- **bandwidth** *ancho-de-banda-en-kbps*
- **no bandwidth** *ancho-de-banda-en-kbps*

CBWFQ asigna un peso a los paquetes derivado de la cantidad de ancho de banda reservado para esa clase. *CBWFQ* empleará ese peso para asegurar que la cola de esa clase se sirve de acuerdo al ancho de banda especificado.

Comando queue-limit:

Para especificar el número máximo de paquetes que la cola puede mantener para la política de una clase, se usará el comando **queue-limit** más un número entre 1 y 64 que especifica el número máximo de paquetes que puede acumular la cola de esa clase (64 por defecto). Para borrar ese límite se empleará la forma **no** de ese comando.

- **queue-limit** *numero-de-paquetes*
- **no queue-limit** *numero-de-paquetes*

Este comando no se puede usar al mismo tiempo que el comando **random-detect** ya que son incompatibles. Utilizar este comando implica que se empleará *Tail Drop* para el descarte de paquetes.

Comando fair-queue (clase por defecto):

Para especificar el número de colas dinámicas reservadas para la clase por defecto se empleará el comando **fair-queue** junto con un valor opcional potencia de 2 en el rango de 16 a 4096 que especifica el número de colas dinámicas. La forma **no** para borrar la configuración anterior.

- **fair-queue** *número-de-colas-dinámicas*
- **no fair-queue** *número-de-colas-dinámicas*

La siguiente tabla incluye el número de colas dinámicas por defecto que *WFQ* y *CBWFQ* (clase por defecto) utilizan cuando se activan sobre una interfaz.

Tabla 3. 13: Número de colas dinámicas por defecto que *WFQ* y *CBWFQ* (clase por defecto) utilizan cuando se activan sobre una interfaz.

Rango de ancho de banda	Numero de colas dinámicas
Menor o igual a 64 kbps	16
Mayor que 64 kbps y menor o igual a 128 kbps	32
Mayor que 128 kbps y menor o igual a 256 kbps	64
Mayor que 256 kbps y menor o igual a 512 kbps	128
Mayor que 512 kbps	256

3.4.10.1.2 Ejemplos

Configurando CBWFQ con Tail Drop

En el siguiente ejemplo configuramos una política llamada cbwfq para que mantenga un ancho de banda asegurado de 2 kbps durante periodos de congestión para los paquetes de una clase denominada c1.

```
Router(config)# policy-map cbwfq
Router(config-pmap)# class c1
Router(config-pmap-c)# bandwidth 2000
Router(config)# interface Serial 3/3
Router(config-if)# service out cbwfq
```

Configurando CBWFQ con Tail Drop y el tamaño máximo de la cola

El mismo ejemplo que antes pero ahora configurando un tamaño de cola de 30 paquetes. En ambos casos se ha utilizado *Tail Drop* como método para descartar los paquetes que excedan el límite de la cola.

```
Router(config)# policy-map cbwfq
Router(config-pmap)# class c1
Router(config-pmap-c)# bandwidth 2000
Router(config-pmap-c)# queue-limit 30
Router(config)# interface Serial 3/3
Router(config-if)# service out cbwfq
```

Configurando CBWFQ con WRED

En el siguiente ejemplo tenemos la misma configuración que en los casos anteriores salvo que se empleará *WRED* como método para el descarte de paquetes.

```
Router(config)# policy-map cbwfq
Router(config-pmap)# class c1
Router(config-pmap-c)# bandwidth 2000
Router(config-pmap-c)# random-detect
Router(config)# interface Serial 3/3
Router(config-if)# service out cbwfq
```

Cuando se configura una política para una clase, especificas su ancho de banda mínimo reservado y asocias la política con una interfaz. *CBWFQ* comprueba si el ancho de banda reservado se puede satisfacer. Si es así, *CBWFQ* reserva una cola para el ancho de banda requerido. Además, cuando se elimina una clase, el ancho de banda disponible para la interfaz se ve incrementado en la cantidad previamente reservada para la clase.

3.4.11 *Diffserv Compliant Weighted Random Early Detection WRED*

Diffserv Compliant Weighted Random Early Detection WRED [25][30][31] permite que *WRED* pueda usar los valores del *DSCP* y el *IP Precedence* cuando calcula la probabilidad de descartar un paquete. Esta característica se puede usar en conjunto con *CBWFQ*. Cuando *WRED* no está configurado el comportamiento del *router* permite rellenar los buffers de salida en periodos de congestión usando la característica *Tail Drop* pero esto tirará muchos paquetes en periodos de congestión debido a la falta de buffer y se utilizará muy poco el enlace cuando no haya congestión. Por contra, *WRED* tira paquetes selectivamente cuando la interfaz de salida comienza a mostrar signos de congestión. Tirando algunos paquetes de manera prematura antes de que los buffers se llenen, *WRED* evita tener que tirar grandes cantidades de paquetes a la vez. De esta forma, *WRED* permite que la línea de transmisión se use completamente todo el tiempo.

Si se configura una clase dentro de un *policy map* para usar *WRED* para el descarte de paquetes en vez de *Tail Drop*, hay que asegurarse de que *WRED* no está configurado sobre la interfaz en la que se tiene intención de asociar esa *service policy*.

Como se ha visto al principio de la sección, se puede configurar *WRED* para que emplee los bits del *IP Precedence* o los bits del *DSCP* a la hora de descartar paquetes. Dado que, los Servicios Diferenciados emplean el valor del *IP DSCP*, sólo se incluirán ejemplos de configuración de *WRED* con dichos bits. Los comandos empleados para configurar *WRED* varían dependiendo si *WRED* se configura a nivel de Interfaz, de clase o de circuito virtual. También se omitirá la configuración a nivel de circuito virtual (para más información dirigirse a la página de Cisco).

Cisco recomienda usar el valor por defecto de 9 para el *exponential weight factor* (ver *WRED* en el capítulo de desarrollo teórico).

3.4.11.1.1 *WRED* a nivel de Interfaz

Si se quiere que *WRED* use el valor *DSCP* cuando calcule la probabilidad de tirar un paquete, se empleará el argumento *dscp-based* con el comando **random-detect** para especificar el valor de *DSCP*.

La siguiente tabla muestra los pasos a seguir para configurar *WRED* en una interfaz:

Tabla 3. 14: Pasos para la Configuración de *WRED* a nivel de Interfaz

	Comando	Propósito
Paso 1	`Router(config)# interface nombre-de-la-interfaz`	Especifica el nombre de la interfaz sobre la que se activará *WRED*.
Paso 2	`Router(config-if)# random-detect dscp-based`	Indica que se empleará *WRED* basado en *DSCP*. De ésta forma *WRED* utilizará el valor de *DSCP* cuando calcule la probabilidad de tirar un paquete.
Paso 3	`Router(config-if)# random-detect dscp valor-dscp umbral-mínimo umbral-máximo [mark-probability-denominator]`	Especifica el umbral mínimo y máximo y, opcionalmente, el denominador del *mark-probability* para el valor de *DSCP* especificado.

3.4.11.1.2 *WRED* a nivel de Clase

Si estamos usando *WRED* a nivel de clase (con *CBWFQ*), los argumentos *dscp-based* y *prec-based* se deben usar dentro de la *policy-map*.

Se usarán los comandos: **policy-map**, **class** y **bandwidth** en este orden. De esta forma se habrá activado *CBWFQ* para esa clase. Entonces, si se quiere que *WRED* use el valor *DSCP* cuando calcule la probabilidad de tirar un paquete, se utilizará el argumento *dscp-based* con el comando **random-detect** para especificar el valor de *DSCP*. Después se empleará el comando **random-detect dscp** para especificar el umbral máximo, mínimo y el denominador de la probabilidad de marcado *(mark-probability denominador)* para el valor de *DSCP*.

Tabla 3. 15: Pasos para la Configuración de *WRED* a nivel de Clase

	Comando	Propósito
Paso 1	`Router(config)# policy-map policy-map`	Especifica el nombre de la política que va a ser creada.
Paso 2	`Router(config-pmap)# class class-map-name`	Especifica el nombre de la clase sobre la que se aplica la política.
Paso 3	`Router(config-pmap-c)# bandwidth {bandwidth-kbps|percent percent}`	Activa *CBWFQ* en la clase, reservando una cantidad de ancho de banda para la clase especificada en kbps o en un porcentaje del total de la interfaz
Paso 4	`Router(config-pmap-c)# random-detect dscp-based`	Indica que se empleará *WRED* basado en *DSCP*. De ésta forma *WRED* utilizará el valor de *DSCP* cuando calcule la probabilidad de tirar un paquete.
Paso 5	`Router(config-pmap-c)# random-detect dscp valor-dscp umbral-mínimo umbral-máximo [mark-probability-denominator]`	Especifica el umbral mínimo y máximo y, opcionalmente, el denominador de la probabilidad de marcado (*mark-probability denominaror)* para el valor de *DSCP* especificado.
Paso 6	`Router(config)# interface nombre-de-la-interfaz`	Especifica el nombre de la interfaz a la que se asociará la política.
Paso 7	`Router(config-if)# service-policy input|output policy-map`	Asignamos la política creada a la interfaz.

En ambas configuraciones (a nivel de interfaz y de clase) si no se le asignan los parámetros de *WRED* a cada valor de *DSCP* mediante el comando **random-detect dscp** *dscp-value min-threshold max-threshold [mark-probability-denominator]*, el Cisco *IOS* asocia cada valor de *DSCP* con unos parámetros de configuración de *WRED* (umbral mínimo, umbral máximo, *mark denominador probability*) por defecto. Esos parámetros también dependen del tamaño de la cola configurada para la clase o la interfaz. Si se aumenta el tamaño de la cola, Cisco actualizará sus tablas y aumentará proporcionalmente los parámetros de *WRED*.

Los valores que el Cisco *IOS* utiliza por defecto son los que aparecen en la siguiente tabla:

Tabla 3. 16: Parámetros de WRED que usa Cisco por defecto.

dscp	Umbral Mínimo	Umbral Máximo	Mark probability
af11	32	40	1/10
af12	28	40	1/10
af13	24	40	1/10
af21	32	40	1/10
af22	28	40	1/10
af23	24	40	1/10
af31	32	40	1/10
af32	28	40	1/10
af33	24	40	1/10
af41	32	40	1/10
af42	28	40	1/10
af43	24	40	1/10

dscp	Umbral Mínimo	Umbral Máximo	Mark probability
cs1	22	40	1/10
cs2	24	40	1/10
cs3	26	40	1/10
cs4	28	40	1/10
cs5	30	40	1/10
cs6	32	40	1/10
cs7	34	40	1/10
ef	36	40	1/10
rsvp	36	40	1/10
default	20	40	1/10

3.4.11.1.3 Ejemplos

WRED a nivel de interfaz para usar los valores del DSCP

El siguiente ejemplo activa *WRED* a nivel de interfaz para que emplee los parámetros de descarte de paquetes (umbral mínimo, umbral máximo, *mark denominador probability*) por defecto que Cisco *IOS* asigna a cada valor de *DSCP*.

Estando en modo de configuración global se emplea el comando **interface serial 0/0** para entrar a la configuración de la interfaz serial y con el comando **random-detect dscp-based**, se activa *WRED* en la interfaz y además se le especifica que debe usar los valores de los *DSCP* a la hora de descartar los paquetes.

```
Router(config)# interface serial 0/0
Router(config-if)# random-detect dscp-based
```

WRED a nivel de clase para usar los valores del DSCP

En el siguiente ejemplo se configura *WRED* para los paquetes de la clase c1 dentro de una política llamada *WRED*. Además se dejará que Cisco asigne los parámetros de *WRED* (umbral mínimo, umbral máximo, *mark denominador probability*) por defecto.

Primero se crea una política llamada *WRED*. Luego se especifica el nombre de la clase sobre la que se va a configurar *WRED*. Con el comando **bandwidth** se reserva un ancho de banda de 2000 bits para la clase. Finalmente con el comando **random-detect dscp-based** se activa *WRED* en la clase. Como no se especifican los parámetros de *WRED*, Cisco *IOS* asignará los que tiene por defecto. Para terminar se asocia la política con la interfaz serial 0/0.

```
Router(config)# policy-map WRED
Router(config-pmap)# class c1
Router(config-pmap-c)# bandwidth 2000
Router(config-pmap-c)# random-detect dscp-based
Router(config)# interface Serial 0/0
Router(config-if)# service out WRED
```

WRED a nivel de clase para usar los valores del DSCP

El siguiente ejemplo es idéntico al anterior salvo que ahora cambiamos los valores por defecto de los parámetros de *WRED* para dos valores de *DSCP*. Para el resto de valores de *DSCP* se mantienen las tablas de parámetros por defecto del Cisco *IOS*.

Con el comando **random-detect dscp 18 40 70 50** establecemos los parámetros de *WRED* para paquetes con un *DSCP* de 18 de la siguiente manera:

- Umbral Mínimo = 40
- Umbral Máximo = 70
- *Mark Denominador Probability* = 50 (Ver explicación de *WRED* apartado 2.3.2.2.2)

Para el valor de *DSCP* 20 también se cambian los parámetros por defecto de *WRED*. Para el resto de valores de *DSCP* se mantendrán los parámetros por defecto que establece la *IOS* de Cisco.

```
Router(config)# policy-map WRED
Router(config-pmap)# class c1
Router(config-pmap-c)# bandwidth 2000
Router(config-pmap-c)# random-detect dscp-based
Router(config-pmap-c)# random-detect dscp 18 40 70 50
Router(config-pmap-c)# random-detect dscp 20 10 40 5
Router(config)# interface Serial 0/0
Router(config-if)# service out WRED
```

3.4.11.1.4 Como Verificar la Configuración de los valores *DSCP*

Los siguientes comandos en el modo de configuración global permitirán examinar las configuraciones:

- **show interfaz** *nombre-de-la-interfaz,* sirve para comprobar si se han aplicado los parámetros de configuración correctamente.

- **show queue** *nombre-de-la-interfaz,* muestra el contenido actual de los paquetes de la cola.

- **show queueing interface** *nombre-de-la-interfaz,* muestra las estadísticas de encolamiento de una interfaz o VC.

- **show policy-map interface** *nombre-de-la-interfaz,* muestra la configuración de las clases configuradas dentro de la política de la interfaz especificada.

3.4.11.1.5 A tener en cuenta

Recordar los siguientes puntos cuando configuremos *WRED*:

- Si usa el argumento *dscp-based, WRED* usará el valor *DSCP* para calcular la probabilidad de tirado.
- Si usa el argumento *prec-based*, *WRED* usará el valor del *IP Precedence* para calcular la probabilidad de tirado.
- Ambos argumentos son mutuamente excluyentes.
- Si no especificas ningún argumento, *WRED* utilizará el valor del *IP Precedence* para calcular la probabilidad de tirar un paquete (el método por defecto).
- El comando **random-detect dscp** se debe usar en conjunto con el comando **random-detect** (*interface*).
- El comando **dscp** se debe usar en conjunto con el comando **random-detect-group**.
- El comando **dscp** se puede usar sólo si utilizas el argumento *dscp-based* con el comando **random-detect-group.**

En resumen, esta herramienta extiende la funcionalidad de *WRED* para soportar los Servicios Diferenciados y *Assured Forwarding (AF) PHB*. Permite a los clientes implementar *AF PHB* mediante el marcado de paquetes de acuerdo a los valores *DSCP* y asignarles probabilidades de tirado preferencial a esos paquetes. Sin olvidar tampoco que sólo se puede usar con paquetes *IP*.

3.4.12 *LLQ*

La herramienta de encolamiento de baja latencia *Low Latency Queueing LLQ* [24][32][34] lleva la disciplina de servicio de colas encolamiento de prioridad estricta *(Strict Priority Queueing)* a *CBWFQ*. *Strict priority Queueing* permite que el tráfico sensible al retardo como el tráfico de voz sea desencolado y enviado antes que paquetes de otras colas, dandole al tráfico sensible al retardo un tratamiento preferente sobre otros tipos de tráfico.

Sin *LLQ*, *CBWFQ* sirve todos los paquetes de manera equitativa basándose en su peso, pero no proporciona prioridad estricta *(strict priority)* para ninguna clase de paquetes. Esto puede suponer un problema para el tráfico de voz que es muy sensible al retardo y especialmente a la varianza del retardo o *jitter*.

LLQ proporciona *Strict Priority Queueing* a *CBWFQ* reduciendo la varianza del retardo *jitter* en las conversaciones de voz. Cuando se activa *LLQ* se emplea una única cola de prioridad estricta *(Strict Priority Queue)* dentro de *CBWFQ* a nivel de clase, permitiendo llevar el tráfico perteneciente a una clase a una *CBWFQ Strict Priority Queue*. Dentro de una *policy map* se pueden configurar más de una clase para que usen *LLQ* pero todo el tráfico de esas clases será encolado dentro de la misma *Strict Priority Queue*.

Una de las diferencias entre el *Strict Priority Queueing* empleado dentro de *CBWFQ* y el que se emplea sin *CBWFQ* está en los parámetros que toma. Fuera de *CBWFQ*, mediante el uso del comando **ip rtp priority**, se da prioridad solamente a los flujos del tráfico de voz que entran por un rango de puertos *UDP*. Mientras que en el *Strict Priority Queueing* empleado dentro de *CBWFQ* se pueden emplear muchos criterios de selección de tráfico como pueden ser: listas de accesso *(access list)*, protocolos, interfaces de entrada e incluso los valores de los *IP DSCP* o *IP Precedence*. Aunque es posible aplicar *LLQ* para varios tipos de aplicaciones de tiempo real, Cisco aconseja emplearlo solamente para el tráfico de voz.

Cuando se configura *LLQ* mediante el comando **priority** para una clase, toma un ancho de banda como argumento que es el ancho de banda máximo en kilobits por segundo (kbps). Este parámetro garantiza un ancho de banda para la clase *priority* pero también acota el flujo de paquetes de esa clase.

Si se produce congestión, cuando el ancho de banda configurado se excede se emplea un algoritmo de *token bucket* para descartar paquetes, midiéndose el tráfico destinado a la cola priority para asegurar que se cumple el ancho de banda configurado para el tráfico de la clase. El tráfico de voz encolado en la cola *priority* es *UDP,* por lo tanto no se adapta al descarte de paquetes realizado por *WRED*. Debido a que *WRED* es ineficiente, no se podrá usar *WRED* (comando **random-detect)** con el comando **priority**. Además, como se emplea un *policing* para descartar paquetes y no hay límites de cola impuestos, el comando **queue-limit** tampoco se podrá usar con el comando **priority**.

Las clases son tratadas por las funciones de *policing* de manera individual. Esto quiere decir que aunque una única *policy map* pueda contener cuatro clases prioritarias y se encolen todas en una única cola prioritaria, se tratan cada una como flujos de tráfico separados.

Esta herramienta será util para gestionar el tráfico de la clase *Expedited Forwarding EF* de los Servicios Diferenciados y darle un tratamiento preferente frente a otros tipos de tráfico como el tráfico de la clase *Assured Forwarding*. Como se veía en la arquitectura definida por el *IETF* para los Servicios Diferenciados, el tráfico de la clase *EF* tendrá requerimientos de poco retardo y poca varianza del retardo *(jitter)*. Cisco *IOS* proporciona una solución a esas necesidades mediante la herramienta *LLQ*.

3.4.12.1.1 Configuración

Usamos el comando **priority** para activar *LLQ*. Los pasos a seguir para configurar *LLQ* dentro de una clase cuando la clase ya está definida previamente son:

Tabla 3. 17: Pasos para la Configuración de *LLQ*

	Comando	Propósito
Paso 1	`Router(config)# policy-map policy-map`	Especifica el nombre de la política que va a ser creada.
Paso 2	`Router(config-pmap)# class class-map-name`	Especifica el nombre de la clase sobre la que se aplica la política.
Paso 3	`Router(config-pmap-c)# priority ancho-de-banda-en-kbps`	Activa *LLQ* dentro de la clase sobre la que estamos configurando la política, reservando una cantidad de ancho de banda para la clase especificada en kbps.
Paso 4	`Router(config)# interface nombre-de-la-interfaz`	Especifica el nombre de la interfaz a la que se asociará la política.
Paso 5	`Router(config-if)# service-policy input|output policy-map`	Asignamos la política creada a la interfaz.

3.4.12.1.2 Ejemplos

Ejemplo de configuración de LLQ dentro de una política

En el siguiente ejemplo, se reserva una cola de prioridad estricta *(Strict Priority Queue)* (con un ancho de banda garantizado de 50 kbps) para el tráfico enviado desde la fuente 10.10.10.10 al destino 10.10.10.20, en el rango de puertos *UDP* del 16384 al 20000 y del 53000 al 56000.

Primero con los comandos siguientes configuramos la lista de acceso 102 para que filtre el tráfico de voz deseado:

```
router(config)# access-list 102 permit udp host 10.10.10.10 host
10.10.10.20 range 16384 20000
router(config)# access-list 102 permit udp host 10.10.10.10 host
10.10.10.20 range 53000 56000
```

Luego, definimos la clase voz, y creamos la política policy1; se reserva una cola de prioridad estricta *(Strict Priority Queue)* para la clase voz, se configura un ancho de banda de 20 kbps para la clase *best-effort* y la clase por defecto la configuramos para que emplee *WFQ*. Por último, se asocia la política con la interfaz serial 0/0.

```
router(config)# class-map voz
router(config-cmap)# match access-group 102

router(config)# policy-map policy1
```

```
router(config-pmap)# class voice
router(config-pmap-c)# priority 50
router(config-pmap)# class bar
router(config-pmap-c)# bandwidth 20
router(config-pmap)# class class-default
router(config-pmap-c)# fair-queue

router(config)# interface serial 0/0
router(config-if)# service-policy output policy1
```

Los paquetes que lleguen a la interfaz serial 0/0 del *router* serán filtrados. Los paquetes cuyo puerto *UDP* se encuentre en el rango del 16384 al 20000 o del 53000 al 56000, que pertenezcan a la fuente 10.10.10.10 y tengan como destino la dirección 10.10.10.20 pasarán a formar parte de la clase "voice". Los paquetes que no cumplan esa condición ni los criterios de selección de la clase "bar" (se han omitido) se clasificarán como pertenecientes a la clase por defecto.

En caso de que se produzca congestión se asegurá un ancho de banda mínimo de 50 kbps para los paquetes de la clase "voice". Los paquetes de esta clase serán desencolados antes que paquetes de la clase "bar" para mantener unas condiciones de retardo y varianza del retardo mínimas. Pero también se aplicará una función de policía mediante un algoritmo de *token bucket* a los paquetes de esa clase para que no sobrepasen el límite establecido.

3.4.12.1.3 Diferencias entre el comando bandwidth y el comando priority

Ambos comandos, **bandwidth** y **priority,** proporcionan un ancho de banda garantizado para los paquetes que cumplen los criterios de selección de una clase de tráfico. Sin embargo, los dos comandos tienen importantes diferencias funcionales que se muestran en la siguiente tabla:

Tabla 3. 18: Diferencias funcionales entre los comandos priority y bandwidth

Función	Comando bandwidth	Comando priority
Ancho de banda mínimo garantizado	Si	Si
Ancho de banda máximo garantizado	No	Si
Tiene un Policer	No	Si
Proporciona baja latencia	No	Si

En periodos de congestión ambos comandos proporcionan un ancho de banda mínimo asegurado para las clases de tráfico. La diferencias está en que el comando **priority** también implementa un ancho de banda máximo garantizado. Internamente, la cola prioritaria emplea un *token bucket* que mide la carga ofrecida y asegura que el flujo de tráfico no excede la tasa configurada. Sólo el tráfico conforme con al *token bucket* se le garantiza baja latencia. Por eso al contrario que ocurre para las clases configuradas con el comando **bandwidth**, las clases que usan el comando **priority** no usan nunca el ancho de banda que sobra, durante periodos de congestión.

Además, los comandos **bandwidth** y **priority** están diseñados para proporcionar diferentes objetivos de *QoS*. La siguiente tabla muestra esos objetivos:

Tabla 3. 19: Diferencias en los Objetivos de *QoS* entre los Comandos priority y bandwidth

Aplicación	Comando bandwidth	Comando priority
Gestión del ancho de banda para enlaces WAN	Si	A veces
Gestión del retardo y de las variaciones del retardo *(jitter)*	No	Si
Mejora del tiempo de respuesta de las aplicaciones	No	Si

La siguiente tabla describe cuando una clase **bandwidth** y una clase **priotity** pueden usar el ancho de banda sobrante:

Tabla 3. 20: Diferencias entre el empleo del Ancho de Banda Sobrante entre los Comandos priority y bandwidth

Comando	Congestión	No Hay Congestión
bandwidth	Se le permite exceder la tasa configuarada	Se le permite exceder la tasa configuarada
priority	Cisco *IOS* clasifica los paquetes y les aplica un medidor de tráfico mediante un token bucket. A los paquetes seleccionados se les aplica una función de policía que los acota a la tasa configurada en bps y los paquetes que exceden la tasa se descartan.	La clase puede exceder su tasa configurada

3.4.13 *CEF*

Cisco express Forwarding (CEF) [33] es una avanzada tecnología de conmutación de la capa *IP*. *CEF* optimiza el rendimiento y la escalabilidad de la red en redes con configuraciones extensas y dinámicas, tales como *Internet*. Redes caracterizadas por intensivas aplicaciones basadas en web o sesiones interactivas. Además *CEF*, como se ha visto, es imprescindible para que funcionen algunas de las herramientas de *DiffServ* explicadas anteriormente.

Los beneficios de usar *CEF* son:

- Mejora el rendimiento: *CEF* carga menos la *CPU* y se puede usar más *CPU* para procesar servicios de la capa 3 tales como *QoS* y encriptación.

- Escalabilidad: *CEF* ofrece capacidad completa de conmutación en cada *line card* cuando el modo distributed *CEF* permanece activo.

- Elasticidad: *CEF* ofrece un nivel sin precedentes de consistencia y estabilidad de conmutación en redes grandes y dinámicas.

El término ***line card*** es un término general que se refiere a un procesador de interfaces que se puede usar en una línea de productos de Cisco. Por ejemplo, un *VIP (Virtual Interface Processor)* es una ***line card*** de la serie 7500 de Cisco.

Consideraciones cuando se implemente *CEF* en la red:

- Los requerimientos mínimos de memoria recomendados en plataformas que lleven información de enrutado completa de *Internet* actualmente son:

  - 128 MB para el procesador de enrutamiento *(route processor)* central.
  - 32 MB para cada *line card*

3.4.13.1.1 Componentes de *CEF*

La información que se almacena convencionalmente en una caché de enrutado, en *CEF* se almacena en varias estructuras de datos. Las estructuras de datos proporcionan unas tablas de consulta optimizadas para el encaminamiento eficiente de paquetes. Los dos componentes principales de *CEF* son:

- *Forwarding Information Base FIB*
- *Adjacency Tables*

3.4.13.1.1.1 Forwarding Information Base

El *FIB* es similar conceptualmente a una tabla de encaminamiento *(routing table)* o base de información *(information base)*. Mantiene una imagen de la información de enrutamiento contenida en la tabla de encaminamiento *IP*. Cuando se producen cambios en la topología o en el enrutamiento de una red, la tabla de encaminamiento *IP* se actualiza y estos cambios se ven reflejados en el *FIB*. El *FIB* mantiene las direcciones del siguiente salto basándose en la información de las tablas de encaminamiento *IP*.

3.4.13.1.1.2 Adjacency Tables

Los nodos de una red se dice que son adyacentes si se pueden alcanzar uno a otro con un único salto. Además de *FIB*, *CEF* utiliza las tablas de adyacencia que mantienen direcciones de salto próximo de la capa de enlace de datos para todas las entradas *FIB*.

Para más información sobre las tablas de adyacencia consultar el documento de *Cisco Express Forwarding*.

3.4.13.1.2 Modos de operación de *CEF*

CEF tiene dos modos de operación:

- Modo de *CEF* centralizado.
- Modo de *CEF* distribuido.

3.4.13.1.2.1 Modo centralizado

Cuando el modo centralizado de *CEF* está activo, el *FIB* y las tablas de adyacencia *(adjency tables)* de *CEF* residen en el *route processor* y el *route processor* ejecuta el encaminamiento expreso. Se puede utilizar este modo cuando las *line cards* no pueden emplear *CEF* o cuando se necesiten emplear herramientas que no son compatibles con el *CEF* distribuido.

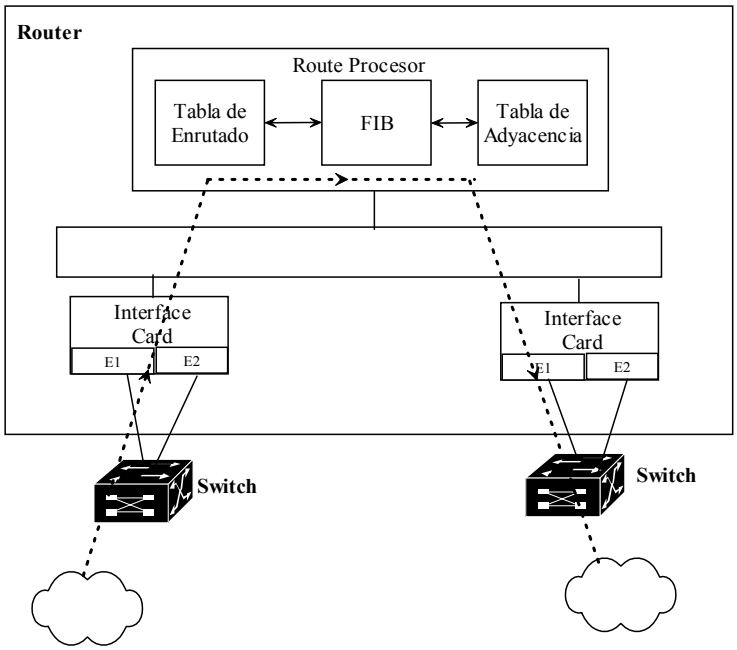

Figura 3. 9 : Modo de Operación Centralizado de *CEF*

3.4.13.1.2.2 Modo Distribuido

Cuando se activa *dCEF*, *line cards* tales como *VIP line cards* mantienen una copia idéntica de las tablas de adyacencia y del *FIB*.

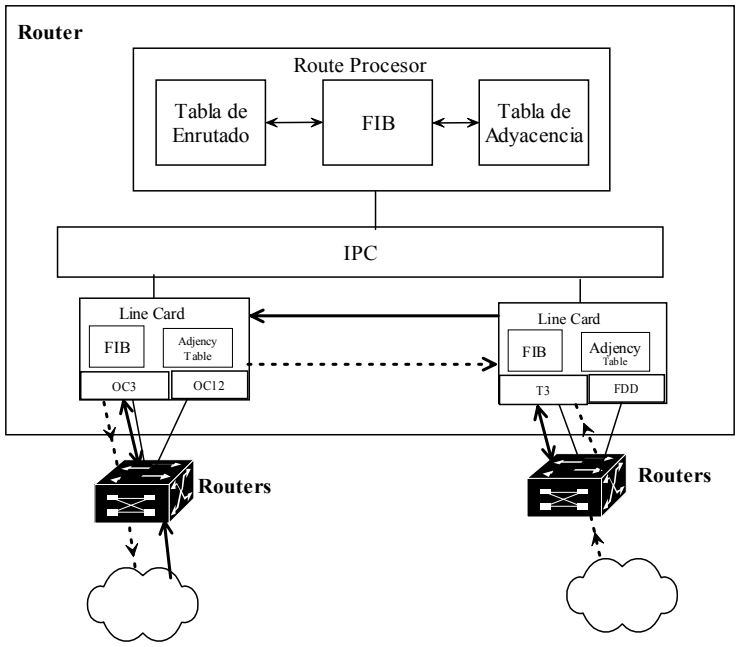

Figura 3. 10: Modo de Operación Centralizado de *CEF*

3.4.13.1.3 Configuración

Hay varios commandos para la configuración de *CEF* o de *dCEF*; comandos para activarlos, para configurar el balance de carga para *CEF*, etc… Aquí se expondrá unicamente el comando de activación de *CEF*, necesario para poder emplear algunas de las herramientas del *Cisco IOS software* para la implementación de los Servicios Diferenciados.

Para la activación del modo de operación estándar de *CEF* se empleará el comando **ip cef switch.** Se usará la forma **no** para desactivarlo.

- **ip cef switch** Activa el modo de operación estándar de *CEF*
- **no ip cef switch** Desactiva el modo de operación estándar de *CEF*

Capítulo 4
Desarrollo Práctico

4.1 Introducción

Se van a llevar a cabo una serie de pruebas cuya finalidad será comprobar el funcionamiento de los *routers* Cisco de la serie 2600 [43] en un entorno de Servicios Diferenciados. Para ello, se verá primero cómo configurar las herramientas que proporcionan estos dispositivos para implementar la arquitectura de Servicios Diferenciados.

En los capítulos anteriores vimos que, en la arquitectura de Servicios Diferenciados había dos tipos de nodos dentro de un dominio *DiffServ*: los nodos interiores y los nodos frontera. Los nodos frontera eran los encargados de acondicionar el tráfico que entraba al dominio y marcarlo con los valores de *DSCP* apropiados. Mientras, los nodos interiores trataban al tráfico dependiendo de esos valores de *DSCP*.

Para las pruebas se dispone de dos *routers* Cisco 2600. Al carecer de un tercer *router* que actúe de nodo interior. Se decide en las pruebas que sólo uno de los *routers* realice la diferenciación de servicios, actuando como nodo frontera, es decir, que en él se lleven a cabo todas las funciones necesarias para implementar los Servicios Diferenciados, en concreto el servicio *Assured Forwarding*: acondicionando el tráfico mediante funciones de policía (con *Token Buckets*), marcándolo, clasificándolo y dándole un tratamiento particular en función de los valores de *DSCP* de cada paquete. El otro *router* sólo servirá de interconexión entre el *router* frontera y los PC's. Esto permitirá crear un cuello de botella en el enlace entre ambos *routers* que provoque congestión y permita observar cómo actúan las diferentes herramientas de *DiffServ* de Cisco frente a ella.

Con el fin de poder comparar cómo el uso del *Assured Forwarding* de *DiffServ* permite garantizar los anchos de banda contratados y estudiar como se realiza el reparto del ancho de banda en exceso, realizamos experimentos en diferentes escenarios con y sin emplear las herramientas que Cisco proporciona para la diferenciación de servicios.

Conviene resaltar que en las pruebas llevadas a cabo, no se diferenciarán los tipos de tráfico por la aplicación a la que pertenecen (protocolo, puerto, etc…), sino que se diferenciarán por la red *LAN* a la que pertenecen. Esto se hace de este modo para comprobar si se pueden emplear los Servicios Diferenciados para asegurar contratos de diferentes clientes.

4.2 Descripción del Entorno de realización de las pruebas

En primer lugar se llevan a cabo diferentes pruebas con dos fuentes generadoras de tráfico *TCP*. Cada una de estas fuentes simula una red *LAN* diferente y las dos están conectadas directamente al *router* (cada una por una interfaz) que implementa los Servicios Diferenciados.

Para la diferenciación de tráficos por red *LAN* se emplearán las listas de control de acceso de los *routers,* que filtraran el tráfico por dirección *IP*. Luego ese tráfico será clasificado en dos clases, una para cada red *LAN*. Si la diferenciación fuera por aplicación el modo de actuar sería el mismo pero las listas de acceso filtrarían el tráfico por protocolo y puerto y no por *IP*.

Como se ha comentado, se creará un cuello de botella en el enlace serie entre los dos *routers*. Esto producirá congestión y se podrá observar como actúan las diferentes herramientas de *DiffServ* para solventarla.

Para acondicionar los flujos de tráfico entrante que llegan a las interfaces del *router* se emplearán mecanismos de *Token Bucket* que comprobarán las características de los flujos y marcarán los paquetes con un *DSCP* u otro, dependiendo de si esos flujos permanecen dentro de los límites contratados o por el contrario exceden esos límites. Resaltar que en estos *Token Buckets* no se descartará paquete alguno, simplemente se marcarán los paquetes. Algunos paquetes si que serán descartados por los gestores y servidores de colas que se encuentren en la entrada del cuello de botella.

Como se ha expuesto, las pruebas van encaminadas a comprobar si, usando los mecanismos que proporciona el software Cisco *IOS*, se garantizan los contratos a las diferentes redes *LAN*. Por esta razón, se supondrá que una de las redes *LAN* tiene contratados 256 Kbps y la otra red variará su contrato para cada prueba desde 256 Kbps hasta 1,768 Mbps, es decir, se irá variando el contrato hasta que la suma de los anchos de banda contratados por las dos redes *LAN* sea igual al ancho de banda del cuello de botella.

Naturalmente, ambas redes intentarán enviar más tráfico que el contratado y los mecanismos de *DiffServ* serán los encargados de asegurar que ambas redes reciben el ancho de banda contratado y de repartir el ancho de banda sobrante de manera equitativa.

Para la generación de tráfico se usarán dos programas: Traffic Generador, TG [44] y Netperf [45]. TG se empleará para generar tráfico *TCP* y Netperf para las pruebas con tráfico *UDP*. La duración de cada una de las pruebas será de 120 segundos. Las fuentes de tráfico *TCP* intentarán mandar tráfico con una tasa exponencial de 2 Mbps, es decir, el ancho de banda del enlace entre los dos *routers*. Las fuentes de tráfico *UDP* generarán tráfico a la tasa máxima de las interfaces de red, aproximadamente unos 10 Mbps. El tamaño de los paquetes será de 1024 bytes.

La siguiente figura resume el entorno expuesto:

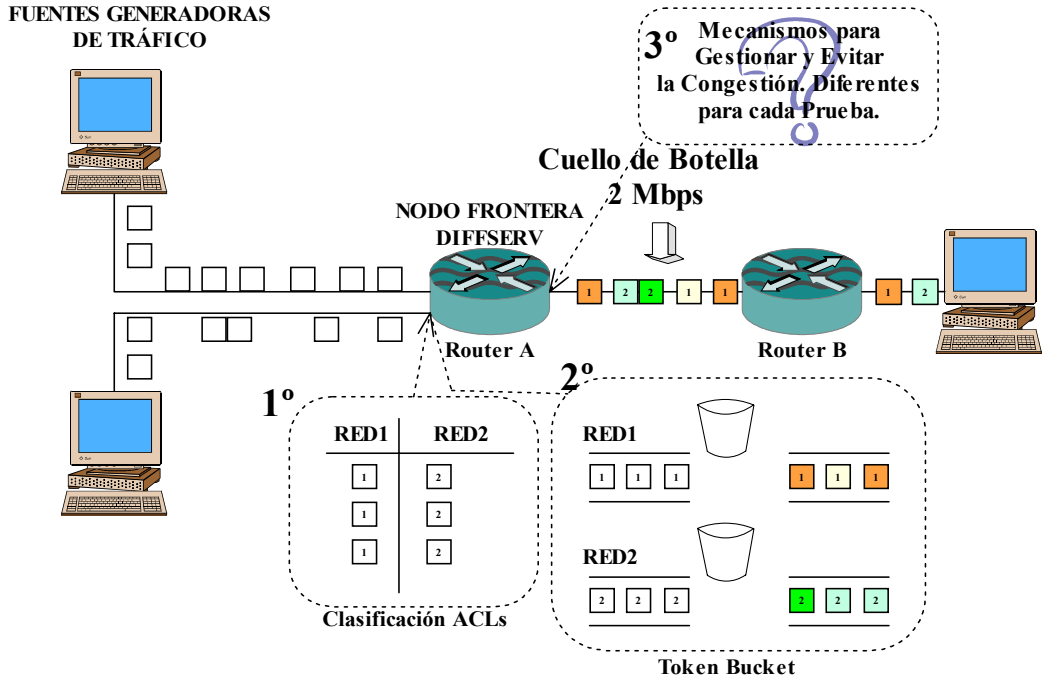

Figura 4. 1: Esquema del Entorno de Realización de las Pruebas

En la figura se aprecian dos fuentes que generan tráfico *TCP* ó *UDP*. Estas dos fuentes simulan cada una, una red *LAN* diferente. En el Router A, que actúa como nodo frontera del dominio de Servicios Diferenciados (notar que detrás de él no hay un dominio de *DiffServ* tal como se define en la arquitectura de *DiffServ*), es donde se configuran todas las herramientas de *DiffServ* de Cisco que se van probar.

En la figura se puede ver también el camino que seguirán los paquetes desde que salen de las fuentes generadoras hasta que llegan a su destino. Al llegar al Router A, los flujos de tráfico se filtran y se clasifican en dos clases, una para cada red *LAN*. Luego, se le aplica al tráfico de cada clase un *Token Bucket* individual. Este *Token Bucket* comprobará si los flujos de tráfico cumplen sus contratos. Los paquetes que cumplan el contrato serán marcados con un *DSCP* y aquellos cuya tasa supere lo contratado se marcarán con un *DSCP* distinto. Por último, los paquetes llegarán a la interfaz serial del Router A, la cuál experimenta congestión debido a que las fuentes de tráfico envían a una tasa mayor que la contratada. En este punto se probarán diferentes mecanismos de gestión y servicio de colas, algunos pertenecientes a diferenciación de servicios y otros no, para gestionar el acceso al enlace serie. El Router B no llevará cabo ninguna acción sobre los paquetes que no sea la de cambiarlos de interfaz.

Como se ha visto, para cada prueba se modificarán los contratos de las fuentes. Para esto simplemente se modificarán los parámetros de configuración de los *Token Buckets*. Las configuraciones que se llevarán a cabo serán:

- Configuraciones sin ningún tipo de Diferenciación de Servicios:
  - Disciplina de Servicio *FIFO* con *Tail Drop*
  - Disciplina de Servicio *FIFO* con *WRED*
  - Disciplina de Servicio *WFQ*
- Configuración con Diferenciación de Servicios:
  - Disciplina de Servicio *FIFO* con *WRED*
  - Disciplina de Servicio *CBWFQ*
  - Disciplina de Servicio *CBWFQ* con *WRED*

Para cada una de ellas se probaran diferentes contratos:

- LAN1: 256 Kbps y *LAN* 2: 256 Kbps
- *LAN* 1: 256 Kbps y *LAN* 2: 512 Kbps
- *LAN* 1: 256 Kbps y *LAN* 2: 768 Kbps
- *LAN* 1: 256 Kbps y *LAN* 2: 1 Mbps
- *LAN* 1: 256 Kbps y *LAN* 2: 1,768 Mbps

4.2.1 Descripción de Equipos

Para realizar las pruebas se dispone de dos *routers* Cisco de la serie 2600 [43] sobre los que aplicaremos las configuraciones necesarias para implementar los Servicios Diferenciados. También se dispone de tres ordenadores que serán las fuentes generadoras de tráfico y medidoras de la *QoS* percibida.

Las series de Cisco 2600 son una familia de routers modulares de acceso multiservicio que proporcionan configuraciones de *LAN* y WAN flexibles, múltiples opciones de seguridad, etc. Este rango de características hacen que la familia Cisco 2600 sea ideal para cubrir las necesidades actuales y futuras de empresas.

Los routers módulares múltiservicio Cisco 2600 ofrecen versatilidad, integración y potencia. Con alrededor de 70 módulos de red e interfaces, la arquitectura modular de la serie Cisco 2600 permite actualizar fácilmente las interfaces para acomodarlas a la expansión de la red.

Las series de routers 2600 de Cisco ofrecen:

- Integración multiservicio de voz y datos
- Acceso a Internet/intranet con firewall
- Acceso a redes privadas virtuales (VPN) con opciones de firewall
- Servicios de acceso telefónico analógico y digital
- Enrutamiento con gestión de ancho de banda
- Enrutamiento entre VLAN

Con un potente procesador RISC y DSP de alto rendimiento y procesadores auxiliares en varias interfaces, la serie Cisco 2600 admite calidad de servicio (*QoS*) avanzada, seguridad y las características de integración en la red.

Características técnicas:

La serie Cisco 2600 proporciona flexibilidad y opciones de densidad de puertos para las distintas LANs. La siguiente tabla resalta algunas de las posibles configuraciones de la serie Cisco 2600:

Tabla 4. 1: Posibles Configuraciones de la Serie Cisco 2600

APLICACIÓN	Nº MÁXIMO ADMITIDO
Llamadas de voz simultáneas	60/4
Conexiones T1/E1 (incluyendo ATM)	8
Modems integrados	16
RDSI PRI (canales B)	64
RDSI BRI	10
Serial asíncrona	37
Serial síncrona	12

- Procesador principal: 80 MHz RISC (Cisco 265x);
- 50 MHz RISC (Cisco 262x); 40 MHz RISC
- (Cisco 261x)
- Memoria Flash: de 8 a 16 MB (Cisco 261x y
- Cisco 262x); de 8 a 32 MB (Cisco 265x sólo)
- Memoria de sistema (DRAM): de 32 a 64 MB

- (Cisco 261x y Cisco 262x); de 32 a 128 MB
- (Cisco 265x sólo, usa SDRAM)
- Ranuras para tarjetas de interfaz WAN: 2
- Ranuras para módulos de red: 1
- Ranura AIM: 1
- Consola/velocidad auxiliar: 115,2 Kbps (máxima)

El *router* A (ver figura 4.2) es un Cisco 2611 con un rendimiento de 15.000 paquetes por segundo (pps) tiene dos interfaces *Ethernet*, un módulo WIC serial de alta velocidad de doble puerto (hasta 8Mbps por puerto). La siguiente figura muestra la parte posterior del Cisco 2611:

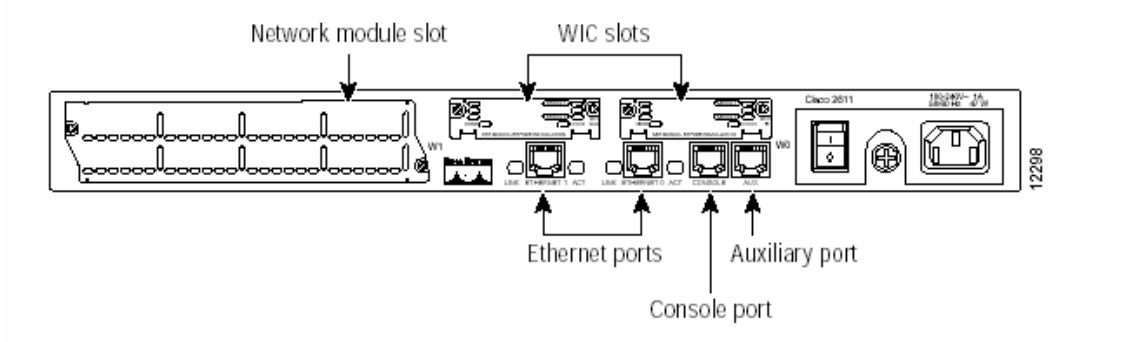

Figura 4. 2: Vista del panel posterior del *router* Cisco 2611

En una de las ranuras WIC tenemos un WIC serial de alta velocidad de doble puerto (hasta 8Mbps por puerto) como se muestra en la figura:

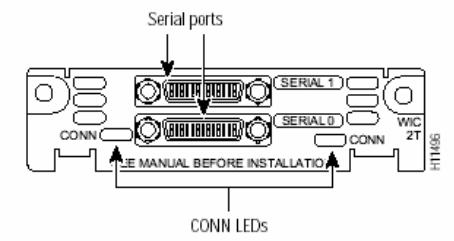

Figura 4. 3: WIC serial de alta velocidad de doble puerto (hasta 8Mbps por puerto)

El *router* B es un *router* Cisco 2620 con un rendimiento de hasta 25.000 pps. Dispone de una interfaz serial WIC 1T de alta velocidad de un puerto, una interfaz *Fast-Ethernet* 10/100 y una interfaz RDSI.

Los PC's tienen un procesador Intel Pentium III a 500Mhz con 128M de memoria RAM. Están equipados con tarjetas de red 10/100.

Los cables para unir los PC's con los *routers* son RJ-45 cat 5 *FTP* cruzados.

Utilizaremos la siguiente topología para realizar las pruebas:

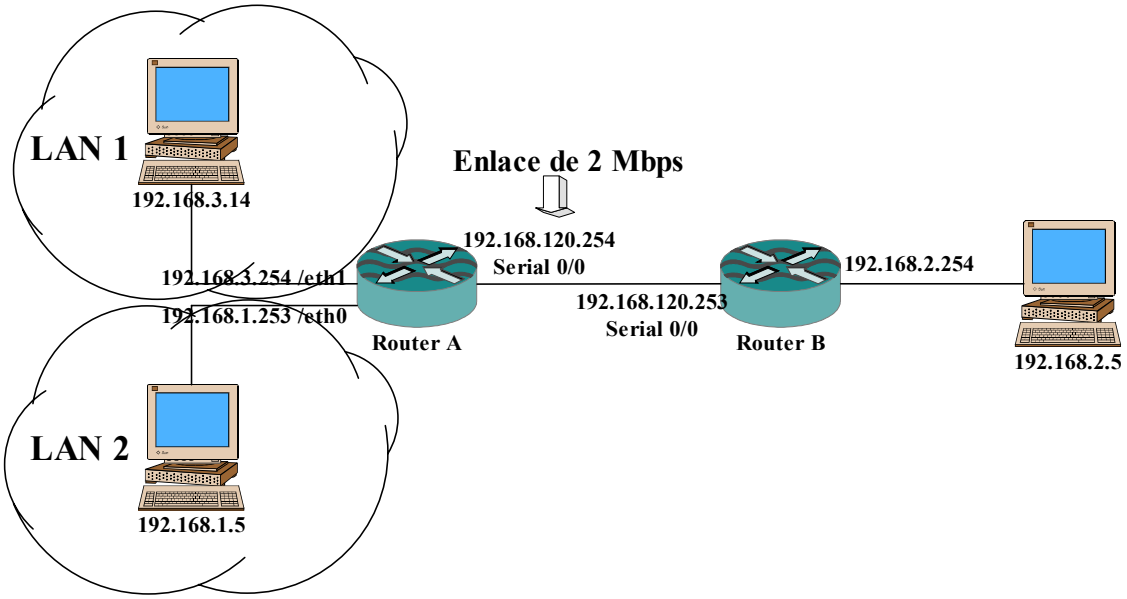

Figura 4. 4: Descripción de las Redes Configuradas

4.2.2 Interconexión de los Equipos

La unión entre ambos *routers* se realiza mediante un cable serie V.35.

Como se vio en el capítulo 3, en enlaces serie un lado debe proporcionar una señal de reloj en el extremo *DCE* de un cable, el otro lado es un *DTE*. Por defecto, los *routers* Cisco son dispositivos *DTE*; sin embargo, en algunos casos se pueden utilizar como dispositivos *DCE*. En la configuración realizada, el *Router* B hará de *DCE* y el *Router* A de *DTE*. Por tanto, el *Router* B será el que debe proporcionar la señal de reloj.

Para la realización de las pruebas se simula un cuello de botella en el enlace serie configurando la velocidad del enlace a 2 Mbps. Las interfaces *Ethernet* tienen una velocidad máxima de transmisión de 10 Mbps y la interfaz *Fast-Ethernet* de 100 Mbps.

Los *PCs* dispondrán de tarjetas de red 10/100. Así, los enlaces entre los *PCs* y el *Router* A serán de 10 Mbps como máximo debido a que las *Ethernet* del *router* sólo alcanzan esa velocidad y el enlace entre el *Router* B y el PC al que está conectado será de 100 Mbps por que en ambos extremos las tarjetas de red alcanzan esas velocidades.

Como se muestra en la figura 4.4, el *PC* con la dirección *IP* 192.168.1.5 se conecta directamente a través de un cable RJ-45 cruzado a la interfaz *Ethernet* 0/0 del *Router* A cuya dirección *IP* será 192.168.1.253. El *PC*, con la dirección *IP* 192.168.3.14, se conecta directamente mediante un cable *RJ-45* cruzado a la interfaz *Ethernet* 0/1 del *Router* A cuya dirección *IP* es 192.168.3.254. El tercer *PC* de dirección *IP* 192.168.2.5 se conectará a la interfaz *Fast-Ethernet* 0/0 del *Router* B de dirección *IP* 192.168.2.254 del mismo modo, mediante el uso de un cable RJ-45 cruzado.

Ambos *routers* están actualizados con la versión del software Cisco *IOS* 12.1. Esta versión incluye mejoras de las características en las áreas de voz, seguridad, interconexionado de redes privadas virtuales (VPN), calidad de servicio *(QoS)*, Multiprotocol Label Switching (MPLS) y multicast.

Todos los *PC's* llevarán el sistema operativo *Linux*. Cada uno de los *PC's* tendrá instalado los programas generadores de tráfico TG y Netperf. Ambos programas servirán para probar las configuraciones creadas.

4.3 Preparación del entorno de realización de las pruebas

Una vez descrita la topología sobre la que se realizarán las pruebas, se va a ver como configurar los *routers* y los *PCs* para crear esa topología.

4.3.1 Configuración Inicial de los Routers

Los routers al principio carecen de configuración inicial, por lo tanto, no estarán configurados ninguno de los módulos de interfaz y será necesario acceder a los routers a través del puerto consola. En este apartado se verá cómo dar esa configuración inicial a los routers A y B cuando se arrancan por vez primera, cómo configurar las interfaces de estos y los protocolos y tablas de enrutamiento necesarios para crear la topología anteriormente expuesta.

4.3.1.1 Configuración *Router* A

PASO1: Para poder acceder por primera vez al *router*, es necesario establecer una sesión de emulación de terminal desde un PC, que estará conectado a través de un puerto de comunicaciones al puerto de consola del *router*. El programa que nos permite establecer la sesión es HyperTerminal, que es una aplicación accesoria en Microsoft Windows.

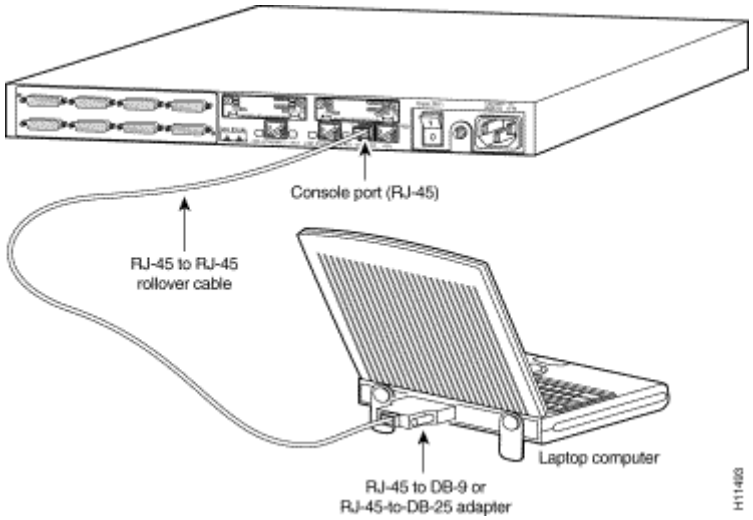

Figura 4. 5: Conexión de PC al Puerto Consola del Router

Los pasos para crear la sesión son:

- Abrir el programa.
- Iniciar nueva conexión. Darle nombre.
- Indicar el puerto de comunicaciones que se está utilizando.
- Paso mas importante: asignar propiedades al puerto de comunicaciones, que son (ver figura 4.6):
    1. Bits por segundo: 9600
    2. Bits de datos: 8

3. Paridad: Ninguna
4. Bits de parada: 1
5. Control de flujo: Ninguno.

- Aceptar.

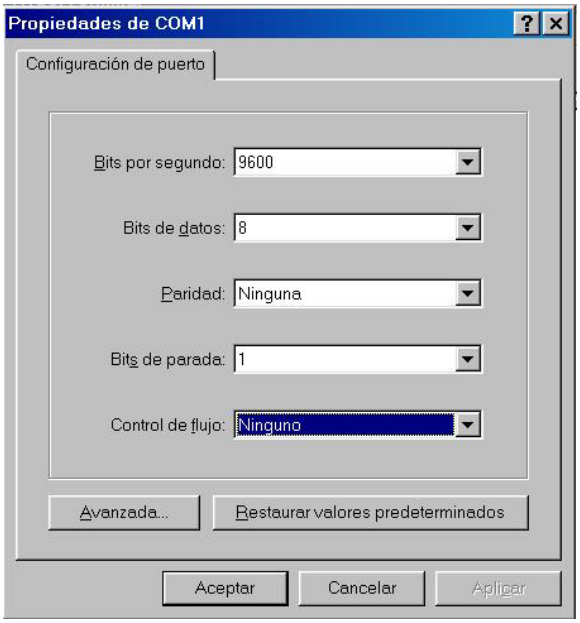

Figura 4. 6: Configuración del Programa HyperTerminal de Microsoft

Tras establecer una sesión, puede desconectarse o volverse a conectar haciendo clic en los iconos de desconexión y conexión. También puede grabar la salida de una sesión seleccionando "Archivo, Grabar". Las sesiones serán grabadas en la carpeta de HyperTerminal con el nombre que le asignó a la sesión y así, no tendrá que volver a crear una nueva sesión.

PASO2: Encender el *Router*.

Cuando el *router* Cisco se pone en marcha, se realizan tres operaciones:

1. El *router* localiza el hardware y lleva a cabo una serie de rutinas de detección del mismo.
2. Una vez que el hardware se muestra en una disposición correcta de funcionamiento, el dispositivo lleva a cabo rutinas de inicio del sistema.
3. Tras cargar el sistema operativo, el dispositivo trata de localizar y aplicar la configuración disponible en la memoria *NVRAM*.

Las configuraciones del *router* se guardan en un tipo especial de memoria llamada memoria de acceso aleatorio no volátil (*NVRAM*). Si no existe ningún archivo de configuración en la *NVRAM*, el sistema operativo lleva a cabo una rutina de configuración inicial basada en preguntas, conocida como diálogo de configuración del sistema. Este modo especial se denomina también diálogo *Setup*. La configuración inicial se utiliza para crear una configuración mínima.

A continuación se detallan los mensajes que aparecen en el programa terminal al arrancar el *router*:
NOTA: el siguiente mensaje varía según el modelo de *router* y sistema operativo de arranque.

```
System Bootstrap, Version 11.3 (1)XA,   PLATFORM   SPECIFIC RELEASE
SOFTWARE (fc1)
Copyright     (c) 1998 by   risen  Systms,    Inc.
C2600 platform with  32768 Kbytes of  main memory
```

<Se omiten los mensajes siguientes>

Muy Importante: No se debe presionar ninguna tecla hasta que no dejen de aparecer mensajes. Si se presionan teclas mientras aparecen mensajes se puede interpretar como que se ha introducido el primer comando.

PASO3: Cuándo aparezca el siguiente mensaje, se debe presionar *Intro* para aceptar la entrada por defecto (yes) que va entre corchetes:

```
Would you like to enter the initial configuration dialog? [yes]: yes
```

NOTA: Si se responde no, se finaliza la auto-instalación y se entra en el modo Cisco *IOS* software CLI.

PASO4: Cuándo aparezca el siguiente mensaje, se debe presionar *Intro* para ver la lista de interfaces disponibles:

```
First, would you like to see the current interface summary?
[yes]:<Intro>
...
```

<Lista de Interfaces Disponibles>

```
...
```

PASO5: Introducir un nombre de *host* para el *router* (introduciremos RouterA):

```
Configuring global parameters:

Enter host name [Router]: RouterA

The enable secret is a password used to protect access to privileged
EXEC and configuration modes. This password, after entered, becomes
encrypted in the configuration.
```

PASO6: Introducir una contraseña enable secret. Esta contraseña está encriptada (más segura) y no se puede ver cuando se mira la configuración. Será la contraseña que el sistema pida para poder entrar a configurar el *router*:

```
Enter enable secret: ********

The enable password is used when you do not specify an enable secret
password, with some older software versions, and some boot images.
```

PASO7: Introducir una contraseña enable que es diferente de la contraseña enable secret. Esta contraseña no está encriptada (menos segura) y se muestra cuando se mira la configuración. Se usa cuando no está la otra contraseña y cuando se usan versiones antiguas del software.

```
Enter enable password: guessme

The virtual terminal password is used to protect access to the router
over a network interface.
```

PASO8: Introduciremos una contraseña virtual terminal, para prevenir de accesos no autorizados al *router* por puertos que no sean el Puerto consola. Por ejemplo, cuando se acceda al *router* mediante Telnet el *router* pedirá esta contraseña.

```
Enter virtual terminal password: guessagain
```

PASO9: Las siguientes preguntas son para la configuración de la red:

```
Configure SNMP Network Management?[yes]: no
```

<Más preguntas sobre los protocolos de red que vamos a configurar >

PASO10: Preguntas sobre las configuraciones de las interfaces. Asignación de IPs, etc… En este paso se configurará únicamente una de las interfaces *Ethernet* para poder acceder al *router* mediante Telnet.

PASO11: Cuando se complete el proceso de configuración para todas las interfaces instaladas en el *router*, se mostrará un script con la configuración creada.

PASO12: Finalmente se mostrará en pantalla tres opciones, donde teclearemos 2 para guardar la configuración en la memoria de acceso aleatorio no volátil NVRAM y salir:

```
[0] Go to the IOS command prompt without saving this config
[1] Return back to the setup without saving this config
[2] Save this configuration to nvram and exit.
```

4.3.1.1.1 Configuración de Interfaces

Configuración de Interfaces *Ethernet*:

Una vez salvada la configuración inicial entramos al *router*. La interfaz *Ethernet* 0/0 se podía haber configurado mediante el diálogo de inicio del *router*. En caso contrario la configuraremos del siguiente modo:

Tabla 4. 2: Configuración de la Interfaz Ethernet 0/0 del Router A

Comando	Descripción
`RouterA>enable`	Pasa a modo EXEC privilegiado
`RouterA> password: *****`	Se debe introducir la contraseña enable secret
`RouterA#configure terminal`	Pasa a modo configuración global
`RouterA(config)#interface Ethernet 0/0`	Pasa a configurar la interfaz indicada
`RouterA(config-if)# ip address 192.168.1.253 255.255.255.0`	Dirección *IP* y máscara
`RouterA(config-if)# half-duplex`	Modo de operación
`RouterA(config-if)#no shutdown`	Dar de alta la interfaz
`RouterA(config-if)#exit`	Salir al modo configuración global

Una vez configurada la interfaz podremos acceder a la configuración del *router* mediante Telnet. Para acceder al *router* mediante Telnet el sistema pedirá la contraseña virtual que se introdujo en el diálogo de configuración inicial del *router*.

Para configurar la interfaz *Ethernet* 0/1 se seguirán los mismos pasos que para configurar la interfaz *Ethernet* 0/0 sólo que en este caso la dirección *IP* será 192.168.3.254.

Configuración de la Interfaz Serial:

Tabla 4. 3: Configuración de la Interfaz Serial 0/0 del Router A

Comando	Descripción
`RouterA>enable`	Pasa a modo EXEC privilegiado
`RouterA> password:*****`	Se debe introducir la contraseña enable secret
`RouterA#configure terminal`	Pasa a modo configuración global
`RouterA(config)#interface serial 0/0`	Pasa a configurar la interfaz indicada
`RouterA(config-if)# ip address 192.168.120.254 255.255.255.0`	Dirección *IP* y máscara
`RouterA(config-if)# encapsulation hdlc`	Tipo de encapsulación
`RouterA(config-if)#no shutdown`	Dar de alta la interfaz
`RouterA(config-if)#exit`	Salir al modo configuración global

No se emplea el comando **clockrate** para indicar la velocidad del enlace porque la interfaz serial del *Router* A hará de *DTE* y por tanto, la interfaz serial del *Router* B será la que determine la velocidad del enlace.

4.3.1.1.2 Configuración del enrutamiento

Tabla 4. 4: Configuración del Encaminamiento del Router A

Comando	Descripción
`RouterA>enable`	Pasa a modo EXEC privilegiado
`RouterA> password: *****`	Se debe introducir la contraseña enable secret
`RouterA#configure terminal`	Pasa a modo configuración global
`RouterA(config)# ip routing`	Habilita el enrutamiento ip
`RouterA(config)# ip classless`	Comando utilizado para que el *router* no descarte paquetes destinados a redes con diferente clase de la perteneciente al *router*. Por defecto, el *router* funciona en modo con clase
`RouterA(config)# ip cef`	Activa *CEF*; necesario para poder configurar algunas de las características de *QoS*

Comando *router* rip:

Para activar el protocolo RIP en el *router* se hará mediante el comando ***router* rip**, después se indicarán las redes a las que el *router* se conecta directamente mediante el comando **network** seguido de la dirección ip de la red.

```
RouterA(config)# router rip
                    network 192.168.1.0
                    network 192.168.3.0
                    network 192.168.120.0
```

Tablas estáticas, gateway por defecto:

Para que el *router* sepa hacia donde dirigir los paquetes para los que no tenga ruta, es decir, aquellos paquetes que vayan a una red diferente de las especificadas en el paso anterios con el comando ***router* rip.** En este caso el *router* encaminará los paquetes por la ethernet 0/0 hacia la dirección ip 192.168.1.254 que es el *router* del laboratorio que tiene acceso a Internet.

```
ip route 0.0.0.0 0.0.0.0 Ethernet0/0 192.168.1.254
```

4.3.2 Configuración del Router B

La inicialización es muy similar a la del *router* A. En el paso 5 introduciremos el nombre de *host* del *router* que será Router B.

4.3.2.1.1 Configuración de Interfaces

Configuración de la Interfaz *Fast-Ethernet*:

Una vez salvada la configuración inicial entramos al *router*. La interfaz *Fast-Ethernet* 0/0 se podía haber configurado mediante el diálogo de inicio del *router*. En caso contrario la configuraremos del siguiente modo:

Tabla 4. 5: Configuración de la Interfaz Fast Ethernet 0/0 del Router B

Comando	Descripción
`RouterA>enable`	Pasa a modo EXEC privilegiado
`RouterA> password: *****`	Se debe introducir la contraseña enable secret
`RouterA#configure terminal`	Pasa a modo configuración global
`RouterA(config)#interface fastEthernet 0/0`	Pasa a configurar la interfaz indicada
`RouterA(config-if)# ip address 192.168.2.254 255.255.255.0`	Dirección *IP* y máscara
`RouterA(config-if)# speed auto`	Configuración de la velocidad de la interfaz
`RouterA(config-if)# half-duplex`	Modo de operación
`RouterA(config-if)#no shutdown`	Dar de alta la interfaz
`RouterA(config-if)#exit`	Salir al modo configuración global

Configuración de la Interfaz Serial:

Tabla 4. 6: Configuración de la Interfaz Serial 0/0 del Router B

Comando	Descripción
`RouterA>enable`	Pasa a modo EXEC privilegiado
`RouterA> password: *****`	Se debe introducir la contraseña enable secret
`RouterA#configure terminal`	Pasa a modo configuración global
`RouterA(config)#interface serial 0/0`	Pasa a configurar la interfaz indicada
`RouterA(config-if)# ip address 192.168.120.253 255.255.255.0`	Dirección *IP* y máscara
`RouterA(config-if)# encapsulation hdlc`	Tipo de encapsulación
`RouterA(config-if)# clockrate 2000000`	Velocidad del enlace a 2 Mbps
`RouterA(config-if)#no shutdown`	Dar de alta la interfaz
`RouterA(config-if)#exit`	Salir al modo configuración global

Aquí si se utiliza el comando **clockrate** ya que la interfaz serial 0/0 del Router B actúa de DCE.

4.3.2.1.2 Configuración del enrutamiento

Tabla 4. 7: Configuración del Encaminamiento del Router B

Comando	Descripción
`RouterA>`**`enable`**	Pasa a modo EXEC privilegiado
`RouterA> password:` **`******`**	Se debe introducir la contraseña enable secret
`RouterA#`**`configure terminal`**	Pasa a modo configuración global
`RouterA(config)#` **`ip routing`**	Habilita el enrutamiento ip
`RouterA(config)#` **`ip classless`**	Comando utilizado para que el *router* no descarte paquetes destinados a redes con diferente clase de la perteneciente al *router*. Por defecto, el *router* funciona en modo con clase
`RouterA(config)#` **`ip cef`**	Activa *CEF*; necesario para poder configurar algunas de las características de *QoS*

Comando *router* rip:

Para activar el protocolo RIP en el *router* se hará mediante el comando ***router* rip**, después se indicarán las redes a las que el *router* se conecta directamente mediante el comando **network** seguido de la dirección ip de la red.

```
RouterA(config)# router rip
                    network 192.168.2.0
                    network 192.168.120.0
```

4.3.3 Guardar la configuración en la NVRAM

Para guardar las configuraciones creadas en la memoria de acceso aleatorio no volátil NVRAM para que al reiniciar el *router* no se pierda la configuración se usará el siguiente comando [20]:

```
RouterA#copy running config startup-config
```

4.3.4 Cambiar la *IOS* al *router*

Será necesario actualizar la imagen del software de Cisco *IOS* [21] ya que la nueva versión tiene herramientas necesarias para implementar los Servicios Diferenciados. La versión que tienen ahora mismo los *routers* es la versión 11.3 y se va actualizar a la versión 12.1.

De la página de Cisco hay que bajarse el programa *TFTP* Server, que será el que sirva para cambiar la imagen del software Cisco *IOS* al *router*, para instalarlo en el *host* en el que tenemos la nueva versión. Este programa servirá tanto para almacenar las nuevas imágenes del software Cisco *IOS* del *router* como para realizar copias de seguridad de las versiones antiguas por si hubiese algún fallo. Para ver la versión actual de la *IOS* usaremos el comando **show versión.**

Pasos preliminares:

1. En la carpeta de archivos del servidor *TFTP* se pondrá la nueva imagen del software de Cisco *IOS*.
2. Se arranca el servidor *TFTP*

Desde el *router*:

3. Verificar que se puede acceder al servidor *TFTP*

RouterA# ping **192.168.1.5**

4. Verificar que el *router* tiene suficiente espacio en la memoria Flash para acomodar la imagen del software Cisco *IOS*:

    RouterA# ***show flash***

    Se comprueba que no hay espacio suficiente para una nueva imagen si no se borra la anterior, por lo tanto, se realizará una copia de seguridad de la imagen que contiene actualmente el *router*.

5. Se realiza la copia de seguridad mediante el comando **copy flash tftp**:

    RouterA# ***copy flash tftp***

    El sistema pregunta el nombre del archivo (por defecto el que tiene):
    Source filename [C2600C113.bin]? ***<intro>***

    El sistema pregunta la dirección donde se encuentra el servidor *TFTP*:
    Address or name of remote host []? ***<192.168.1.5>***

    El sistema pide el nombre del archivo de destino (por defecto el que ya tenía):
    Destination filename [C2611C113.bin]? ***<intro>***

    Confirmación de que el proceso se está llevando a cabo:
    !!!!!!!!!!!!!!.!!!.!!
    !!!!!!!!!.!!!!!
    !!!!!!!!!!!!!!!!!!!.!!!
    !!!!!!!!!!!!!!!
    <….>

6. Borrado de la memoria flash:

    RouterA# ***erase flash***

    Pide confirmación (presionar intro):
    Erasing the flash filesystem will remove all files! Continue? [confirm] ***<intro>***

    Confirmación de que el proceso se está llevando a cabo:
    Erasing device... eeeeeeeeeeeeeeeeeeeeeeeeeeeeeeee ...erasedee
    Erase of flash: complete

7. Cargamos la nueva *IOS* en la memoria flash:

    *RouterA#****copy tftp flash***

    El sistema pregunta la dirección donde se encuentra el servidor *TFTP*:
    Address or name of remote host [192.168.1.5]? ***<intro>***

    El sistema pregunta el nombre del archivo (por defecto el único que hay):
    Source filename [c2600-i-mz.121-5.T7.bin]? ***<intro>***

    El sistema pide el nombre del archivo de destino (por defecto el que ya tenía):
    Destination filename [c2600-i-mz.121-5.T7.bin]? ***<intro>***

    Confirmación de que el proceso se está llevando a cabo:
    Loading c2600-i-mz.121-5.T7.bin from 192.168.1.5 (via Ethernet0/0): !!!!!!!!!!!!

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!!!!!!!!
<….>
[OK - 5112988/10225664 bytes]

Verifying checksum...  OK (0xA108)
5112988 bytes copied in 83.701 secs (61602 bytes/sec)
```

En la figura 4.7 se ve el servidor de *TFTP*:

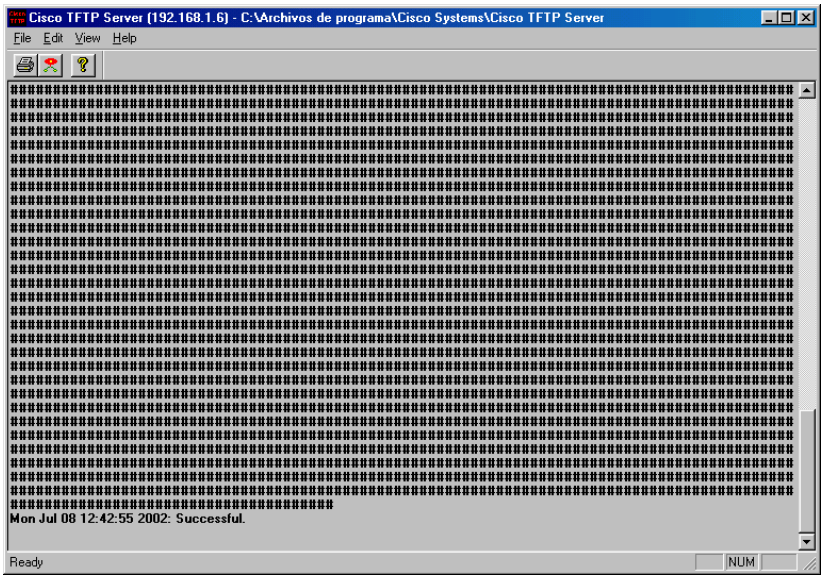

Figura 4. 7: Captura de Pantalla del Programa Servidor *TFTP*

8. Se reinicia el *router*:

```
RouterA# reload
```

Las configuraciones de los *routers* son;

RouterA

```
RouterA#show config

Building configuration...

Current configuration : 1911 bytes
!
version 12.1
no service single-slot-reload-enable
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname RouterA
!
logging rate-limit console 10 except errors
enable secret 5 $1$z8Hk$oxkZfOgx.Xze7bwmn11hQ/
enable password cartagena
!
ip subnet-zero
ip flow-cache timeout inactive 60
ip flow-cache timeout active 1
ip cef
```

```
!
!
no ip finger
!
!
interface Ethernet0/0
 ip address 192.168.1.253 255.255.255.0
full-duplex
!
interface Serial0/0
 ip address 192.168.120.254 255.255.255.0
 no keepalive
!
interface Ethernet0/1
 ip address 192.168.3.254 255.255.255.0
 full-duplex
!
interface Serial0/1
 no ip address
 no ip mroute-cache
 shutdown
!
router rip
 network 192.168.1.0
 network 192.168.3.0
 network 192.168.120.0
!
ip classless
ip route 0.0.0.0 0.0.0.0 Ethernet0/0 192.168.1.254
ip http server
!
access-list 101 permit tcp host 192.168.3.14 host 192.168.2.5
access-list 101 permit udp host 192.168.3.14 host 192.168.2.5
access-list 102 permit tcp host 192.168.1.5 host 192.168.2.5
access-list 102 permit udp host 192.168.1.5 host 192.168.2.5
!
line con 0
 transport input none
line aux 0
line vty 0 4
 password cisco2600
 login
!
no scheduler allocate
end
```

RouterB

```
RouterB#show conf

Using 1210 out of 29688 bytes
!
version 12.1
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
!
hostname RouterB
!
enable secret 5 $1$O690$4xocu3txpMFltAnpnYK.j.
enable password cartagena
!

!
ip subnet-zero
!
```

```
isdn voice-call-failure 0
!
!
!
!
!
interface FastEthernet0/0
 ip address 192.168.2.254 255.255.255.0
 no ip directed-broadcast
 no ip mroute-cache
 speed auto
 half-duplex
!
interface BRI0/0
 no ip address
 no ip directed-broadcast
 no ip mroute-cache
 shutdown
 isdn guard-timer 0 on-expiry accept
!
interface Serial0/0
 ip address 192.168.120.253 255.255.255.0
 no ip directed-broadcast
 no ip mroute-cache
 no keepalive
 clockrate 2000000
!
router rip
 network 192.168.2.0
 network 192.168.120.0
!
ip classless
ip route 0.0.0.0 0.0.0.0 Serial0/0 192.168.120.254
no ip http server
!
!
line con 0
 transport input none
line aux 0
line vty 0 4
 password cisco2600
 login
!
end
```

4.3.5 Configuración de los *PC's*

4.3.5.1 Configuración de red:

En cuanto a los PC, simplemente tenemos que configurar las direcciones *IP* de las tarjetas de red de cada uno de ellos. Como todos los *PC's* tienen como sistema operativo la versión 8.0 de la distribución de *Linux* Suse, se utilizará la herramienta Yast 2 para configurar el entorno de red de cada uno de los *PC's*:

El *PC* con la dirección 192.168.1.5 tendrá como puerta de enlace por defecto la dirección 192.168.1.253.

El PC con la dirección 192.168.3.14 tendrá como puerta de enlace por defecto la dirección 192.168.3.254.

El PC con la dirección 192.168.2.5 tendrá como puerta de enlace por defecto la dirección 192.168.2.254.

4.3.5.2 Instalación de Netperf y TG

Para la realización de las pruebas se han empleado los generadores de tráfico TG [44] y Netperf [45].

Netperf es una herramienta de test (benchmark) que se puede usar para medir el rendimiento de muchos tipos de interconexionados diferentes. Proporciona tests para comprobar el throughput unidireccional y la latencia de extremo a extremo. Entre los entornos que se pueden medir usando netperf se encuentran *TCP* y *UDP*. Por otro lado, TG es un generador de tráfico *TCP* y *UDP*. Puede generar tráfico *TCP* o *UDP* constante, uniforme, exponencial, etc. En cuanto a la instalación de ambos programas, TG no requiere instalación alguna, simplemente copiar los archivos del directorio 'bin' en una máquina Linux. Para instalar netperf seguiremos los siguientes pasos:

1. Como usuario privilegiado crear el directorio netperf dentro de /opt; **mkdir /opt/netperf**

2. Descomprimir los archivos con el comando **tar –xvzf**

3. Compilar con el comando **make**

4. Hacer **make install**

5. Editar el fichero /etc/services y añadirle la línea; **netperf 12865/tcp**

6. Editar el fichero /etc/inetd.conf y añadirle la línea; **netperf stream tcp nowait root /opt/netperf/netserver netserver**

Una vez instalados los programas se explicará sin entrar en detalle como ejecutarlos.

Para ejecutar TG se necesita un PC que funcione de cliente y un PC que funcione de servidor. El PC cliente será el que genere el tráfico y el programa servidor se encargará de recibirlo y de medir la tasa con la que llega. El cliente TG utiliza dos archivos cuando arranca: uno con los parámetros de la prueba a realizar y otro donde se almacenarán los resultados medidos en el cliente. Del mismo modo, el servidor TG emplea dos archivos: uno con los parámetros del servidor como el número de puerto donde escuchará y otro donde almacenará los resultados de la prueba. Posteriormente, se utilizará una utilidad llamada gengraph.pl para unir los resultados del cliente y del servidor y generar las gráficas. Esta utilidad crea archivos de datos para varios programas generadores de gráficas.

El archivo de configuración del cliente de TG contiene lo siguiente:

```
on 0:2 tcp 192.168.2.5.4323
average bandwidth 2000000
mtu 1514
arrival exponential 0.004096 length constant  1024
time 120
```

Con esta configuración los clientes se esperán 2 segundos para establecer la conexión con el servidor de la dirección 192.168.2.5 y puerto 4323. Se le indica el ancho de banda medio de tráfico que se quiere enviar, el tamaño de la trama (mtu 1514) y que genere paquetes de longitud constante (1024) a un intervalo de tiempo de 0.004096 segundos (2 Mbps) con tasa exponencial. La duración de la prueba será de 120 segundos.

Y el archivo del servidor:

```
on 0:2 tcp 192.168.2.5.4323 server
at 1.1 wait 240
```

Con esta configuración el servidor comenzará a escuchar peticiones de los clientes 2 segundos después de ejecutarse. Se le indican la máquina y el puerto donde escuchará las peticiones de los clientes. Después de 240 segundos el servidor terminará (wait 240).

En las pruebas habrá dos clientes (192.168.3.14 y 192.168.1.5) y dos servidores (los dos en 192.168.2.5). Ya que hay dos servidores en la misma máquina se configurarán para escuchar a los clientes en puertos distintos.

La sintaxis es la siguiente:

Tabla 4. 8: Comandos de Ejecución de TG

Comando	Descripción
./tg -f –i tcp_server.tg –o servidor.txt	Ejecuta TG en la parte del cliente con los parámetros indicados por el archivo tcp_server.tg y almacena los resultados en el archivo servidor.txt
./tg -f –i tcp_client.tg –o cliente.txt	Ejecuta TG en la parte del servidor con los parámetros indicados por el archivo xxxx.tg y almacena los resultados en el archivo xxxxx.

Para ejecutar Netperf una vez que está activado como servicio en la máquina servidor, lo único que hay que hacer es ejecutar netperf en la máquina cliente con los parámetros necesarios dependiendo de la prueba.

En las configuraciones llevadas a cabo en las que una fuente genera tráfico UDP y la otra TCP se usará netperf para generar ambos tipos de tráfico. Los parámetros que se emplearán serán:

Tabla 4. 9: Comandos de Ejecución de Netperf

Comando	Descripción
./netperf –H dirección_ip_servidor **–l** tiempo_prueba	Ejecuta el cliente de Netperf para que se conecte al servidor situado en la dirección ip introducida como parámetro después de **–H** y durante un tiempo indicado después de **-l** (Tráfico *TCP*)
./netperf –t *UDP*_STREAM –H dirección_ip_servidor **–l** tiempo_prueba -- -m tamaño_del_paquete	Ejecuta el cliente de Netperf para que se conecte al servidor situado en la dirección ip introducida como parámetro después de **–H** y durante un tiempo indicado después de **-l** (Tráfico *UDP*)

Para conocer más información sobre estos programas de generación de tráfico y testeo de conexiones ver las referencias [44],[45] y [46].

4.4 Configuraciones Específicas

Se ha visto cómo crear configuraciones generales para preparar el entorno de realización de las pruebas. En ellas se han inicializado los *routers*, se han configurado las interfaces y se han activado los protocolos de encaminamiento.

En los siguientes apartados se explicará cómo crear las configuraciones específicas de los routers para llevar a cabo cada una de las pruebas. Primero se verá la creación de las listas de control de acceso y las clases de tráfico que se emplearán en todas las pruebas así como un ejemplo de creación de los Traffic *Policing* que medirán y marcarán el tráfico de las redes. Después, las configuraciones específicas de cada prueba se expondrán en sus apartados correspondientes.

4.4.1 Creación de las listas de control de acceso

Estas listas permitirán filtrar el tráfico que llega a las interfaces del *router* para poder clasificarlo posteriormente.

Desde el modo privilegiado, se accede al modo de configuración global para crear las listas de acceso. Para ver más información sobre la creación de las listas de acceso dirigirse al anexo A.

```
RouterA# configure terminal
RouterA(config)# access-list 101 permit tcp host 192.168.3.14 host
192.168.2.5
RouterA(config)# access-list 101 permit udp host 192.168.3.14 host
192.168.2.5
RouterA(config)# access-list 102 permit tcp host 192.168.1.5 host
192.168.2.5
RouterA(config)# access-list 102 permit udp host 192.168.1.5 host
192.168.2.5
```

Con esto se han creado dos listas de acceso: una para filtrar el tráfico *TCP* o *UDP* de origen el *host* 192.168.3.14 y con destino 192.168.2.5 y otra para filtrar el tráfico *TCP* o *UDP* de origen el *host* 192.168.1.5 y con destino 192.168.2.5.

4.4.2 Creación de las clases de tráfico

Todo lo relativo a las clases de tráfico se explicó en el apartado del *MQC* del capítulo 3. Desde el modo privilegiado, se accede al modo de configuración global para crear las clases.

```
RouterA# configure terminal
RouterA(config)# class-map match-all trafico3.14-2.5
RouterA(config-cmap)# match access-group 101
RouterA(config-cmap)# exit
RouterA(config)# class-map match-all trafico1.5-2.5
RouterA(config-cmap)# match access-group 102
RouterA(config-cmap)# exit
```

Se han creado dos clases, una para el tráfico perteneciente al *host* 192.168.3.14 y de destino 192.168.2.5 y otra para el tráfico perteneciente al *host* 192.168.1.5 y de destino 192.168.2.5.

4.4.2.1 Configuración de los *Traffic Policing* (modificación de los contratos)

Se tendrán que crear dos policy-maps diferentes ya que hay dos interfaces distintas por las que entra el tráfico (*Ethernet* 0/0 y 0/1). Para ver como se crean políticas de *policing* remitirse al apartado de Traffic *Policing* del capítulo 3.

```
RouterA# configure terminal
RouterA(config)# policy-map TrafficPolicing3.14_2.5
RouterA(config-pmap)# class trafico3.14-2.5
```

```
RouterA(config-pmap-c)# police 256000 48000 96000 conform-action set-
dscp-transmit 18 exceed-action set-dscp-transmit 20
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# exit
RouterA(config)# policy-map TrafficPolicing1.5_2.5
RouterA(config-pmap)# class trafico1.5-2.5
RouterA(config-pmap-c)# police 256000 48000 96000 conform-action set-
dscp-transmit 18 exceed-action set-dscp-transmit 20
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# exit
```

Se asocian las políticas con las interfaces por las que entra el tráfico:

```
RouterA(config)# interface Ethernet 0/0
RouterA(config-if)# service-policy input TrafficPolicing1.5_2.5
RouterA(config-if)# exit
RouterA(config)# interface Ethernet 0/1
RouterA(config-if)# service-policy input TrafficPolicing3.14_2.5
RouterA(config-if)# exit
```

La configuración llevada a cabo será para aquellos casos en los que los paquetes de ambas redes se marquen de igual forma, distinguiendo únicamente si son IN u OUT. Estas políticas marcarán todos los paquetes conformes con el contrato de ambas redes con el *DSCP* 18 y todos los paquetes de ambas redes que excedan su respectivo contrato con el *DSCP* 20.

Esta configuración es válida para el caso de que los contratos de las dos redes sean de 256 Kbps, para el resto de configuraciones en las que uno de los contratos varía, los parámetros para cada una de las pruebas serán:

Tabla 4. 10: Parámetros de los Traffic *Policing* según los Contratos

Prueba	Host	Contrato	Política	Parámetros		
				Tasa Media	Normal Burst	Exceded Burst
1	192.168.3.14	256 Kbps	TrafficPolicing3.14_2.5	256000	48000	96000
	192.168.1.5	256 Kbps	TrafficPolicing1.5_2.5	256000	48000	96000
2	192.168.3.14	256 Kbps	TrafficPolicing3.14_2.5	256000	48000	96000
	192.168.1.5	512 Kbps	TrafficPolicing1.5_2.5	512000	96000	192000
3	192.168.3.14	256 Kbps	TrafficPolicing3.14_2.5	256000	48000	96000
	192.168.1.5	768 Kbps	TrafficPolicing1.5_2.5	768000	144000	288000
4	192.168.3.14	256 Kbps	TrafficPolicing3.14_2.5	256000	48000	96000
	192.168.1.5	1 Mbps	TrafficPolicing1.5_2.5	1024000	192000	384000
5	192.168.3.14	256 Kbps	TrafficPolicing3.14_2.5	256000	48000	96000
	192.168.1.5	1,768 Mbps	TrafficPolicing1.5_2.5	1768000	331500	663000

Estos parámetros serán válidos para todas las configuraciones excepto para las de *WRED* con doble perfil de descarte, en ellas se aplicarán algunos cambios. El resto de la configuración del *router* depende del caso en el que se esté. En los siguientes apartados se irán viendo esas configuraciones específicas.

4.5 Configuraciones sin ningún tipo de diferenciación de servicios

Para estos experimentos disponemos de la configuración vista anteriormente. Ninguno de los dos *routers* utilizará mecanismo alguno de diferenciación, es decir, independientemente de

cómo se realice el marcado de paquetes todos ellos irán a parar a colas que tratarán a todos los paquetes por igual. El tráfico será generado por dos fuentes *TCP* que generarán tráfico con una tasa exponencial de 2 Mbps (utilizando el programa TG). Mientras que la LAN1 tiene un contrato fijo de 256 Kbps, la LAN2 variará su contrato desde 256 Kbps hasta 1,768 Mbps. También se llevarán a cabo pruebas en las que una fuente ofrece tráfico *TCP* a la red y la otra ofrece tráfico *UDP* (con el programa Netperf) para ver como afecta el tráfico no 'colaborador' en estas configuraciones.

Las configuraciones que se probarán son:

- Disciplina de Servicio *FIFO* con *Tail Drop*
- Disciplina de Servicio *FIFO* con *WRED* (en este caso actúa como *RED*)
- Disciplina de Servicio *WFQ*

4.5.1 Disciplina de Servicio de Colas *FIFO* con *Tail Drop*

Recordando lo explicado en el apartado 2.3 sobre los mecanismos de Gestión y Control activo de la congestión, se expuso que:

a) El encolamiento de primero en entrar, primero en salir *(First-in, first-out FIFO),* es la disciplina para servir paquetes más básica, todos los paquetes se tratan igual, colocándolos en una única cola y sirviéndolos en el mismo orden en el que fueron colocados en la misma. *FIFO* se denomina también primero en llegar, primero en servirse *(First-come,first-served FCFS)*.

b) *Tail Drop* significa la ausencia completa de un gestor de la memoria de la cola. Cuando un paquete llega al final de una cola completamente llena. El paquete se descarta al igual que todos los que lleguen tras él hasta que hay espacio disponible en la cola.

De lo que se deduce que los paquetes irán llenando la cola *FIFO* en periodos de congestión y cuando se llegue al límite de la memoria de la cola, los nuevos paquetes que lleguen serán descartados. Como se vio, esto puede provocar:

a) La sincronización global de las fuentes de tráfico *TCP*, es decir, que todas comiencen o dejen de transmitir al mismo tiempo. Dando lugar a periodos en los que el enlace se encuentra desaprovechado seguidos de otros en los que en enlace está congestionado. Produciéndose muchas pérdidas.

b) Flujos de tráfico *UDP* que no saben cuando hay congestión acaparan todo el ancho de banda del enlace.

4.5.1.1 Configuraciones

Las listas de acceso, las clases y las políticas de *policing* son las mismas que en caso general. Lo único que se debe configurar en este caso es la cola *FIFO* que habrá en la interfaz serial 0/0. La interfaz serial tiene como disciplina de servicio de colas por defecto a *WFQ*. Para cambiarla se entrará en el modo de configuración de la interfaz y empleará el comando **no fair-queue**, con este comando se deshabilitará la cola *WFQ* establecida por defecto y se activará *FIFO*. En el *router* se hará lo siguiente:

```
RouterA# configure terminal
RouterA(config)# interface serial 0/0
```

```
RouterA(config-if)# no fair-queue
RouterA(config-if)# exit
```

El siguiente esquema muestra el proceso que se llevará a cabo con los paquetes que ingresen en el Router A:

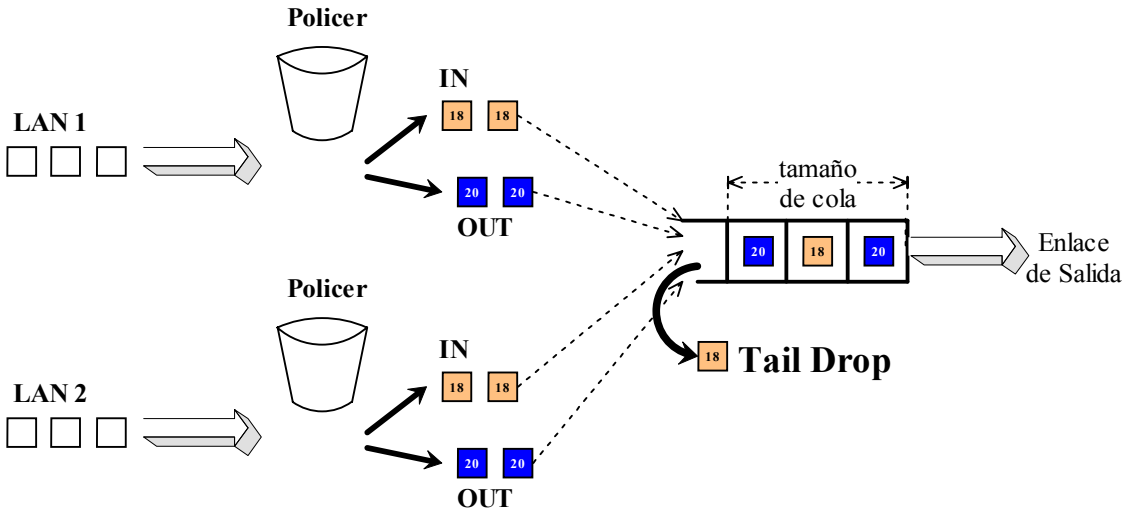

Figura 4. 8: Esquema del Funcionamiento de la Configuración de la cola *FIFO* con Tail Drop

Los paquetes llegan a las interfaces del Router A y son filtrados mediante las listas de acceso 101 y 102 configuradas. Los paquetes se clasifican en dos clases diferentes una para cada red *LAN*. Los paquetes de cada clase son medidos y marcados según cumplan o no el contrato mediante un *Token Bucket*. Todos los paquetes de ambas clases conformes con el contrato se marcan con el *DSCP* 18 y todos los que excedan el contrato se marcan con el *DSCP* 20. Una vez marcados, los paquetes se envían a una cola *FIFO* si se produce congestión en el enlace de salida y allí serán descartados por *Tail Drop* si se produce desbordamiento de la cola.

4.5.1.2 Resultados y Comentarios

En la siguiente tabla se muestran los resultados de las cinco pruebas llevadas a cabo. La primera columna indica la fuente de la que proviene el tráfico y su contrato. La segunda columna llamada "tráfico IN" muestra el tráfico que el *Token Bucket* ha marcado como conforme por respetar el contrato. La tercera columna llamada "tráfico OUT" muestra el tráfico que el *router* marca como fuera del contrato. Finalmente la última columna llamada "tráfico total" muestra el tráfico total de cada una de las fuentes que llega al Router A durante los 120 segundos de duración de la prueba. Los resultados vienen dados Kbits/seg.

Como se ha visto en la configuración: los paquetes se filtrarán mediante las *ACLs* y se clasificarán en dos clases, una para cada red. Posteriormente, pasarán a través de un mecanismo de *Token Bucket* que se encargará de marcarlos. Finalmente, llegarán a la interfaz serial 0/0 del Router A donde serán encolados en una cola *FIFO* en caso de congestión y posiblemente descartados por *Tail Drop* si se produce desbordamiento de la cola. Estos resultados los proporciona el Router usando los comandos **show policy-map interface *Ethernet* 0/0** y **show policy-map interface *Ethernet* 0/1.**

Tabla 4. 11: Resultados del *Token Bucket* (Cola *FIFO* con *Tail Drop*)

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	258	769	1028
Host B: 256 K	259	755	1014

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	252	893	1155
Host B: 512 K	536	418	954
Host A: 256 K	264	690	954
Host B: 768 K	791	337	1128
Host A: 256 K	261	819	1080
Host B: 1 M	966	34	1000
Host A: 256 K	267	228	495
Host B: 1,768 M	1651	0	1651

En la tabla 4.11 se muestran los paquetes descartados a causa de la congestión en la cola *FIFO* de la interfaz serial 0/0 del Router A. Estos resultados los proporciona el Router usando los comandos **show interface serial 0/0.**

Tabla 4. 12: Paquetes Descartados en cola *FIFO* usando *Tail Drop*

Prueba	Paquetes Descartados
1	43
2	50
3	33
4	33
5	40

De los resultados que se muestran en la Tabla 4. 12, se observa que los contratos se garantizan en todos los casos excepto en el último. Como se explicó anteriormente, una cola *FIFO* sirve los paquetes de tal manera que se reparte el ancho de banda de manera más o menos equitativa entre ambas redes. Por esta razón, el caudal total de ambas redes en todas las pruebas excepto en la última, es aproximadamente el mismo (1M para cada red, que es igual al 50% del enlace serie).

Donde se ven diferencias más significativas es en cuanto al reparto del ancho de banda en exceso. Se aprecia que el *host* A se queda con más porcentaje del ancho de banda sobrante. Incluso en el último caso, en el que no debería de haber tráfico en exceso ya que la suma de los contratos de ambas fuentes será igual a 2 M (total del enlace serie), el *host* A obtiene 227 Kbps llegando a impedir que se cumpla el contrato para el *host* B.

Con los resultados obtenidos se podía pensar que usando disciplinas de colas *FIFO* se pueden garantizar los contratos, por lo menos, para los primeros casos. Pero hay que tener en cuenta que las pruebas se han realizado usando tráfico *TCP*. Cuando se produce congestión, *Tail Drop* comienza a descartar paquetes de la cola *FIFO*. Estos descartes son una manera implícita de avisar a las fuentes de tráfico *TCP* que disminuyan su ventana de transmisión y por tanto la tasa de transmisión. Por todo esto, ambas conexiones consumen más o menos la misma cantidad de ancho de banda del enlace y no se producen excesivos descartes, a pesar de que los programas generadores intentarán llegar hasta la tasa de 2 Mbps.

Pero donde se aprecia claramente que este tipo de disciplina no permite garantizar contratos, es cuando alguna de las dos fuentes emplea un tipo de tráfico no 'colaborador' como *UDP*. Cuando se producen descartes de paquetes en la cola *FIFO* debido a la congestión, *UDP* no tiene ningún método para saber de esas pérdidas y por tanto continuará transmitiendo a la tasa máxima. En el caso de que a una misma cola lleguen paquetes *UDP* y *TCP*, si se produce congestión, *TCP* reducirá su tasa de transmisión para no seguir congestionando al *router* pero *UDP* no, aprovechándose de que *TCP* transmite menos para introducir más paquetes en el enlace. Esto provocará que *TCP* apenas consiga ancho de banda del enlace sea cual sea su contrato.

Para ver como el tráfico *UDP* afecta a los contratos veamos los siguientes resultados. En la Tabla 4. 13, la última columna muestra los valores del caudal alcanzado en cada prueba, es decir, el tráfico *UDP* o *TCP* que consigue llegar a su destino. En las pruebas correspondientes a los resultados de la Tabla 4. 13, el *host* A tiene contratados 256 Kbps para ambas pruebas y el *host* B tiene contratados 1 Mbps para la primera prueba que aparece y 1,768 Mbps para la segunda prueba. El *host* A será una fuente de tráfico *UDP* mientras que el *host* B será una fuente de tráfico *TCP*.

Tabla 4. 13: Resultados del *Token Bucket* (Cola *FIFO* con *Tail Drop*) para tráfico *UDP y TCP*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	251	9219	9470	1.95
Host B: 1 M TCP	32	4	36	0.03
Host A: 256 K UDP	251	9219	9470	1.95
Host B: 1,768 M TCP	32	0	32	0.02

En la siguiente tabla se muestran los paquetes descartados a causa de la congestión en la cola *FIFO* de la interfaz serial 0/0 del Router A:

Tabla 4. 14: Paquetes Descartados en cola *FIFO* usando *Tail Drop* con tráfico *UDP y TCP*

Prueba	Paquetes Descartados
1	109441
2	109423

Se puede apreciar claramente que el tráfico *UDP* se queda con prácticamente todo el ancho de banda del enlace, aunque su contrato sea mucho menor. Por tanto, con la disciplina de cola *FIFO* usando *Tail Drop* no se garantizan los anchos de banda contratados si todo el tráfico se introduce en la misma cola.

4.5.2 Disciplina de Servicio *FIFO* con *WRED*

La siguiente configuración es una cola *FIFO* con *WRED* como método para el descarte de paquetes de la cola. Si se recuerda lo expuesto en el Capítulo 2 sobre la disciplina de gestión de la memoria de cola *WRED*, había dos tipos de *WRED*: un *WRED* con un único perfil de descarte para todos los paquetes de una cola y un *WRED* con varios perfiles de descarte para los paquetes de una misma cola según fueran estos paquetes IN o OUT. En la siguiente configuración se empleará *WRED* con un solo perfil de descarte para todos los paquetes de la cola, pertenezcan éstos a una red *LAN* o a otra y respeten o no los contratos preestablecidos. De esta forma, aunque al entrar al Router A los paquetes se clasifiquen según pertenezcan a la red *LAN* 1 o a la 2 y se marquen como IN o como OUT dependiendo de si cumplen o no sus contratos, todos irán a parar a una misma cola *FIFO* con un único perfil de descarte (en realidad son dos pero tienen los mismos valores) en caso de congestión.

Independientemente de que los *Token Bucket* para los paquetes de ambas redes se realicen por separado, los paquetes se marcarán con los mismos *DSCP*, es decir, todos los paquetes IN de ambas redes se marcarán con el valor de *DSCP* 18 y todos los paquetes OUT de ambas redes se marcarán con el valor de *DSCP* 20.

Cuando se sobrepase el umbral mínimo, *WRED* comenzará a descartar algunos paquetes y cuando se sobrepase el umbral máximo *WRED* descartará todos los paquetes nuevos que lleguen a la cola.

En la siguiente figura aparece un esquema del recorrido de los paquetes a través del Router A.

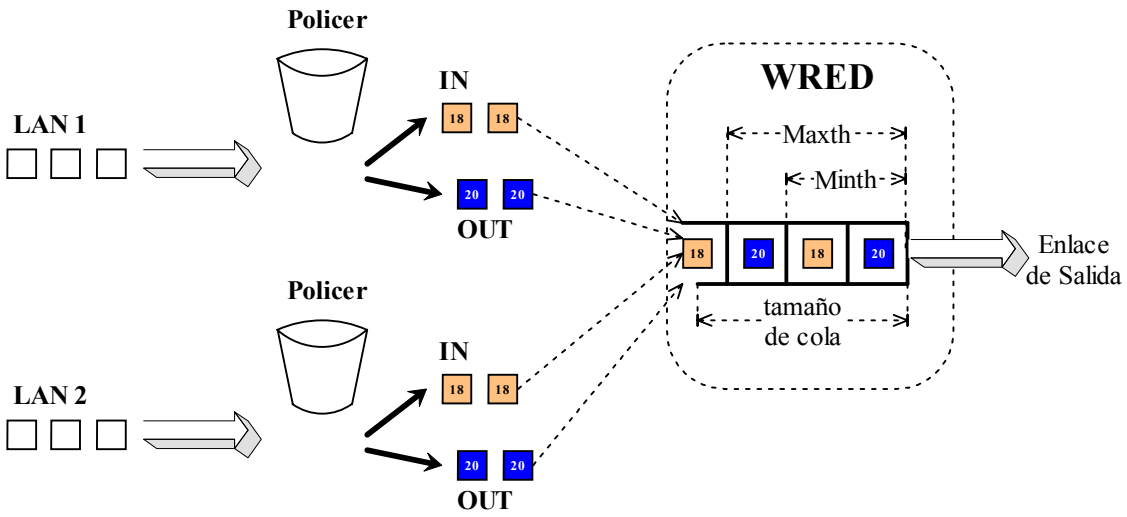

Figura 4. 9: Esquema del Funcionamiento de la Configuración de la cola *FIFO* con *WRED*

4.5.2.1 Configuraciones

Las listas de acceso, las clases y las políticas de *policing* son las mismas que en caso general. Lo único que se debe configurar en este caso es *WRED* en vez de *Tail Drop* como método de descarte de paquetes sobre la interfaz de salida serial 0/0. En el *router* se hará lo siguiente:

```
RouterA# configure terminal
RouterA(config)# interface serial 0/0
RouterA(config-if)# random-detect dscp-based
RouterA(config-if)# random-detect dscp 18 24 40 10
RouterA(config-if)# random-detect dscp 20 24 40 10
RouterA(config-if)# exit
```

Con esto hemos activado *WRED* a nivel de interfaz para que use los parámetros asociados a los valores de *DSCP* a la hora de descartar paquetes. Como en este caso sólo le llegarán al *router* paquetes marcados con los valores de *DSCP* 18 y 20 sólo se configurarán los parámetros de *WRED* para esos valores. El resto de parámetros que *WRED* tiene para cada uno de los valores de *DSCP* se dejarán a sus valores por defecto. Notar que se configuran los mismos perfiles de descarte para el tráfico IN y para el tráfico OUT de ambas redes.

4.5.2.2 Resultados y Comentarios

La tabla de resultados que se muestra a continuación contiene las mismas columnas que en ejemplo anterior por lo tanto no se volverán a explicar. Estos resultados los proporciona el Router usando los comandos **show policy-map interface *Ethernet* 0/0** y **show policy-map interface *Ethernet* 0/1.**

Tabla 4. 15: Resultados del *Token Bucket* (Cola *FIFO* con *WRED*)

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	261	805	1066
Host B: 256 K	263	736	969
Host A: 256 K	261	709	970
Host B: 512 K	523	575	1099
Host A: 256 K	261	745	1007
Host B: 768 K	783	278	1061
Host A: 256 K	261	804	1065
Host B: 1 M	974	26	1000
Host A: 256 K	260	822	1082
Host B: 1,768 M	984	0	984

El tráfico IN es el tráfico de valor *DSCP* 18 y el tráfico OUT es el tráfico con valor de *DSCP* 20. Recuérdese que los paquetes son marcados al entrar al Router A; los paquetes que cumplen el contrato se marcan con el valor de *DSCP* 18 y los que sobrepasan el contrato se marcan con el valor de *DSCP* de 20. En la Tabla 4. 15 se observan prácticamente los mismos resultados que para la cola *FIFO* con *Tail Drop*, es decir, que los contratos se cumplen para todos las pruebas excepto para la última. El *host* cuyo contrato es menor en las pruebas, se lleva un mayor porcentaje del ancho de banda en exceso. Incluso en la última prueba que no debería de llevarse nada, se lleva 8192 Kbps impidiendo que se cumpla el contrato para el *host* B.

En la siguiente tabla se muestran los paquetes descartados mediante *WRED* a causa de la congestión en la cola de la interfaz serial 0/0 del Router A. Estos resultados los proporciona el *router* usando el comando **show queueing interface serial 0/0.**

Tabla 4. 16: Paquetes Descartados en cola *FIFO* usando *WRED*

Prueba	TRAFICO IN (paquetes)	TRAFICO OUT (paquetes)
1	23	42
2	27	44
3	38	35
4	33	25
5	37	27

Al igual que en el caso de la cola *FIFO*, la prueba de que empleando solamente una cola con *WRED* con un solo perfil de descarte no se pueden garantizar de manera satisfactoria los contratos, la dan los resultados usando tráfico *UDP*.

En la tabla siguiente se muestran los resultados obtenidos para dos pruebas diferentes; una en la que el contrato es de 256 Kbps para ambas redes *LAN* y otra en el que una red *LAN* tiene contratados 256 Kbps y la otra 768 Kbps.

Tabla 4. 17: Resultados del *Token Bucket* (Cola *FIFO* con *WRED*) para tráfico *UDP y TCP*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	251	9219	9470	1.95
Host B: 256 K TCP	13	8	21	0.01
Host A: 256 K UDP	251	9219	9470	1.95
Host B: 768 K TCP	19	0	19	0.01

En esta tabla se puede apreciar claramente la imposibilidad de garantizar el cumplimiento de los contratos usando esta configuración. Se observa que la fuente de tráfico *UDP* se queda con todo el ancho de banda del enlace ya que colapsa la cola con sus paquetes, provocando que las fuentes *TCP* reduzcan sus ventanas de transmisión enormemente. Los valores de la última columna indican el caudal total que llega al *host* de destino.

En la siguiente tabla se pueden ver los paquetes descartados por *WRED* para las pruebas de la tabla anterior. La columna "random drop" muestra los paquetes que se descartan usado *WRED* y la columna "*Tail Drop*" indica los paquetes descartados por desbordamiento de la cola. Se puede apreciar que se tiran muchos más paquetes del tráfico OUT, lo cual es normal ya que el *Token Bucket* no realiza descarte alguno y la mayoría de los paquetes *UDP* sobrepasaran el límite de lo contratado. Pero como no hay distinción entre los paquetes *UDP* o *TCP*, *WRED* descartará indistintamente paquetes de uno u otro protocolo, saliendo muy perjudicado el flujo *TCP* que reduce su ventana de transmisión al darse cuenta de los descartes.

Tabla 4. 18: Paquetes Descartados en cola *FIFO* usando *WRED* con tráfico *UDP y TCP*

Prueba	TRÁFICO IN		TRÁFICO OUT	
	Random Drop	***Tail Drop***	***Random Drop***	***Tail Drop***
1	82/86592	2804/2978496	3061/3232416	103459/109254944
2	86/91264	2795/2961824	3011/3179616	103482/109276992

4.5.3 Disciplina de Servicio *WFQ*

De nuevo, recordando lo expuesto en el Capítulo 2 sobre las disciplinas de servicio de colas, se tenía que *FQ* está diseñado para asegurar que cada flujo tenga un acceso justo a los recursos de la red y evita que un flujo de ráfagas consuma más ancho de banda que la parte que le corresponde. En *FQ*, primero el sistema clasifica los paquetes en flujos y los asigna a una cola dedicada especialmente para ese flujo. Las colas se sirven siguiendo un tiempo en orden *round-robin*.

Los beneficios de *FQ* son:

- El primer beneficio de *FQ* es que un flujo con demasiadas ráfagas o un flujo que no colabore no degradarán la calidad de servicio que reciban otros flujos debido a que se aísla a cada flujo en su propia cola.

- Si un flujo intenta consumir más de su ancho de banda, esto sólo afectará a su cola y por lo tanto no influirá en la ejecución de las otras colas.

Por otro lado, *WFQ* es la base de un tipo de disciplinas para servir colas diseñadas para solucionar las limitaciones del modelo *FQ*:

- *WFQ* soporta flujos con diferentes requerimientos de ancho de banda. Esto lo logra dándole a cada cola un peso que le asigna un porcentaje diferente del ancho de banda de salida.

- *WFQ* también soporta paquetes de longitud variable de forma que los flujos con paquetes mayores no dispongan de un ancho de banda mayor que los flujos cuyos paquetes sean de menor tamaño.

Después de esta breve explicación, se va a configurar el *router* para emplear *WFQ* como disciplina de servicio de colas en la interfaz serial 0/0, que como se sabe está conectada

directamente con el enlace de 2 M donde se produce la congestión. También se realizarán las pruebas oportunas y se comentarán los resultados.

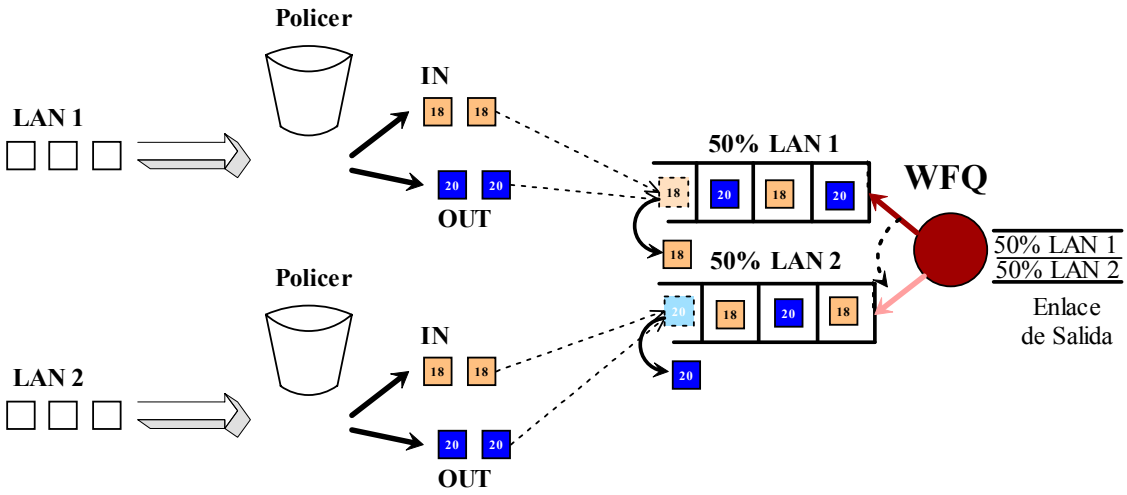

Figura 4. 10: Esquema del Funcionamiento de la Configuración de WFQ

4.5.3.1 Configuraciones

Las listas de acceso, las clases y las políticas de *policing* son las mismas que en caso general. Por lo tanto, lo único que tendrá que hacer en este caso es activar *WFQ* en la interfaz serial 0/0. *WFQ* viene activado por defecto en esta interfaz pero si no fuera así el comando para activarlo es **fair-queue**. En el *router* se hará lo siguiente:

```
RouterA# configure terminal
RouterA(config)# interface serial 0/0
RouterA(config-if)# fair-queue
RouterA(config-if)# exit
```

4.5.3.2 Resultados y Comentarios

En la siguiente tabla se muestran los resultados de las pruebas realizadas. En ella se pueden apreciar resultados parecidos a los casos anteriores en los que se podían garantizar los contratos para todos los casos a excepción del último de ellos. Se puede ver también que se lleva más cantidad de tráfico en exceso la fuente de menor contrato.

Aunque también se aprecia que el caudal total está mucho más igualado que en los casos anteriores. Esto se debe fundamentalmente al propio funcionamiento de *WFQ* en el que los flujos de tráfico se separan en colas diferentes y se sirven en orden round-robin de manera equitativa. Este reparto equitativo se apreciará mejor cuando se realicen pruebas con tráfico *UDP*.

Tabla 4. 19: Resultados del *Token Bucket*, Disciplina *WFQ*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	261	769	1030
Host B: 256 K	262	772	1034
Host A: 256 K	261	769	1030
Host B: 512 K	522	511	1034
Host A: 256 K	261	770	1031
Host B: 768 K	784	250	1034

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	261	770	1031
Host B: 1 M	1020	15	1034
Host A: 256 K	261	769	1030
Host B: 1,768 M	1035	0	1035

En la siguiente tabla se muestran los paquetes descartados en las colas de la interfaz serial 0/0 en la se está utilizando *WFQ* como disciplina de servicio de colas. Estos resultados los muestra el *router* usando el comando **show queueing interfaz serial 0/0.**

Tabla 4. 20: Paquetes descartados en colas *WFQ*

Prueba	Paquetes Descartados
1	26
2	26
3	32
4	28
5	31

Donde se aprecian diferencias notables respecto a las configuraciones anteriores es en los siguientes resultados. Ahora hay una fuente que genera tráfico *UDP* que es el *host* A, el cual tiene contratados 256 Kbps para ambas pruebas. El *host* B será una fuente de tráfico *TCP* que variará su contrato de 256 Kbps en la primera prueba, hasta 1,768 Mbps en la segunda.

En estas pruebas se observa que se pueden garantizar los contratos cuando estos no excedan el 50% del ancho de banda del enlace de salida (1 M), ya que si lo exceden, como ocurre en el caso de la segunda prueba donde el *host* B tiene contratados 1,768 M (el 80% del enlace) el contrato no se podrá garantizar para esta fuente. Esto se debe fundamentalmente al funcionamiento de *WFQ* en el que se reserva una cola para cada flujo de tráfico y se sirven los paquetes de la misma en orden round-robin. Así, independientemente del tipo de tráfico, el ancho de banda del canal de salida se reparte al 50% entre las dos redes *LAN*.

Tabla 4. 21: Resultados del *Token Bucket*, Disciplina *WFQ* con *UDP y TCP*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	251	9218	9469	0.98
Host B: 256 K TCP	258	753	1011	0.97
Host A: 256 K UDP	251	9219	9470	0.98
Host B: 1,768 M TCP	1010	0	1010	0.97

Por todo ello, aunque evidentemente *WFQ* funciona mucho mejor que las configuraciones anteriores usando *Tail Drop* o *WRED* para todo tipo de tráfico. Con ninguna de estas configuraciones se ha conseguido garantizar los contratos cuando alguno de ellos supera el 50% del ancho de banda del enlace de salida.

En las configuraciones siguientes se emplearán las herramientas de Cisco para ver si así se logran garantizar los contratos independientemente de su tamaño y del tipo de tráfico que envíen las redes.

4.6 Configuraciones con diferenciación de servicios

Como se ha comprobado con las pruebas anteriores, es imposible con las configuraciones realizadas garantizar contratos que superen el 50% de la capacidad disponible o repartir de manera equitativa el ancho de banda en exceso. Además solamente con *WFQ* se podían garantizar los contratos cuando las fuentes generaban tráfico *UDP*, siempre y cuando fuera inferior al 50% del ancho de banda del enlace claro.

En las configuraciones con diferenciación de servicios, se emplearán las herramientas que ofrece Cisco para implementar el servicio *Assured Forwarding* de *DiffServ*, se llevarán a cabo pruebas y se comentarán los resultados obtenidos.

La topología será la misma que en las configuraciones, es decir, primero se clasifican los paquetes, luego se les aplica un *Token Bucket* y se marcan y por último llegan a la interfaz congestionada donde se les aplicará un tratamiento u otro dependiendo de la herramienta de *DiffServ* configurada.

Para este caso, también se llevarán a cabo tres configuraciones diferentes que serán:

- *WRED* con varios perfiles de descarte para cada tipo de tráfico.
- *CBWFQ*
- *CBWFQ* con *WRED*

4.6.1 *WRED*

La topología será prácticamente la misma que para la configuración anterior de *WRED*, pero en este caso se configurarán dos perfiles de descarte diferentes para los paquetes de la cola. Así, para los paquetes que los *Token Bucket* marquen como IN los parámetros de *WRED* serán; un umbral mínimo de 40, un umbral máximo de 70 y un *mark probability denominador* de 50.Y para los paquetes marcados como OUT por el *Token Bucket* los parámetros de *WRED* serán; un umbral mínimo de 10, un umbral máximo de 40 y un *mark probability denominador* de 5. Todo esto quiere decir que se comenzarán a tirar antes los paquetes OUT que los IN dentro de la cola *FIFO*. Hay que decir que el tamaño máximo de la cola es de tamaño 200 aunque el umbral máximo de *WRED* sea de 70 paquetes.

Un esquema de la topología sería el siguiente:

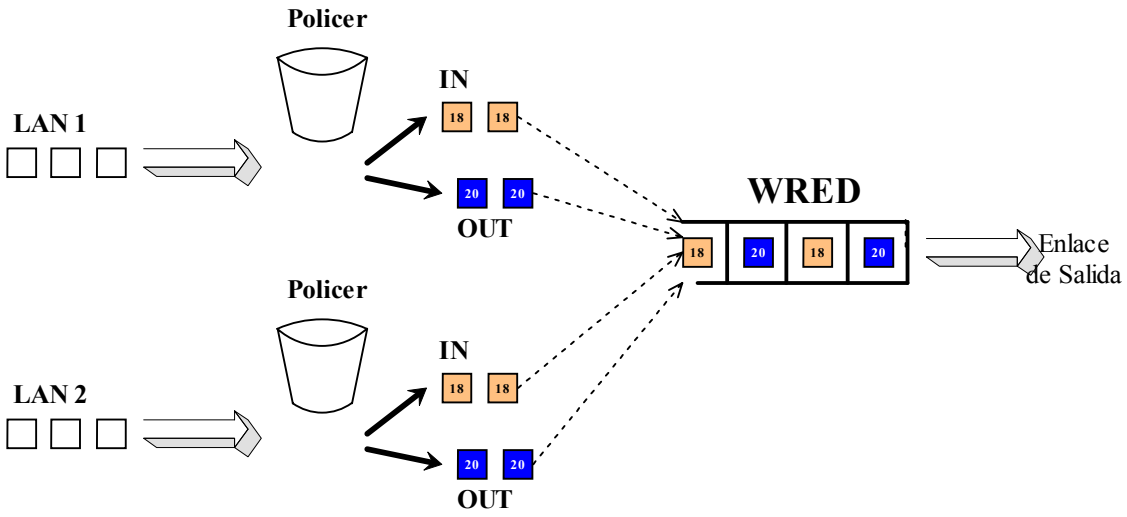

Figura 4. 11: Esquema del Funcionamiento de la Configuración de *WRED* de *DiffServ*

En el esquema anterior se puede apreciar que el funcionamiento es muy parecido a las configuraciones anteriores, esto es, los paquetes se clasifican, se marcan a través de un *Token Bucket* y se envían a una cola *FIFO* con *WRED*. Al igual que en los casos anteriores, todos los paquetes que cumplen el contrato se marcan como IN y todos los paquetes que exceden el contrato se marcan como OUT, independientemente de cuál sea su fuente origen.

La diferencia, como se ha comentado, está en que en esta ocasión la cola *FIFO* tiene dos perfiles de descarte diferentes para cada tipo de tráfico. Para entender esto utilizaremos la siguiente figura:

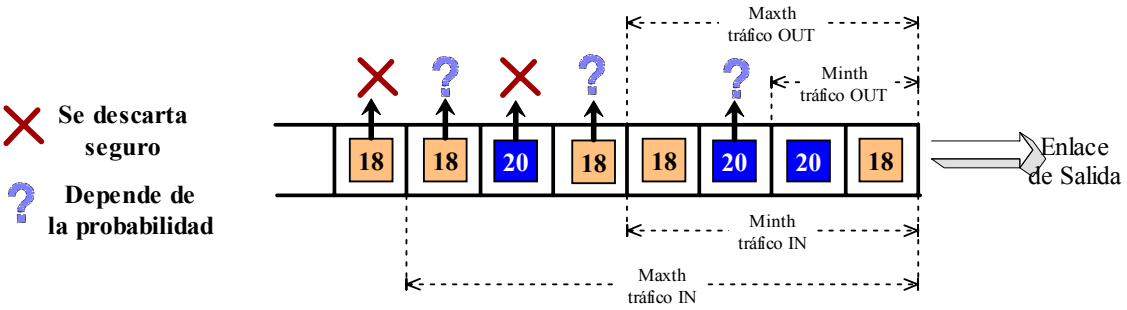

Figura 4. 12: Funcionamiento de *WRED*

En la figura anterior, se muestra el tratamiento que llevará a cabo *WRED* sobre los paquetes que ingresen en la cola debido a la congestión. El funcionamiento de *WRED* se explica en detalle en el capítulo 2.

4.6.1.1 Configuraciones

Las listas de acceso, las clases y las políticas de *policing* son las mismas que en caso general. Por tanto, lo único que se deberá activar y configurar en este caso es *WRED* a nivel de interfaz con los parámetros adecuados. En el *router* se hará lo siguiente:

```
RouterA# configure terminal
RouterA(config)# interface serial 0/0
RouterA(config-if)# hold-queue 200 out
RouterA(config-if)# random-detect dscp-based
```

```
RouterA(config-if)# random-detect dscp 18 40 70 50
RouterA(config-if)# random-detect dscp 20 10 40 5
RouterA(config-if)# exit
```

Mediante el comando **hold-queue 200 out** se establece el tamaño de la cola en 200. Luego se activa *WRED* a nivel de interfaz y se asocian los parámetros adecuados a cada valor de *DSCP*.

4.6.1.2 Resultados y Comentarios

En la siguiente tabla se muestran los resultados obtenidos para las cinco pruebas realizadas. En ella se puede observar que ahora sí se garantizan los contratos aunque sobrepasen estos el 50% del ancho de banda del enlace. Se ve por otro lado que el ancho de banda en exceso sigue sin repartirse al 50% entre las dos fuentes al igual que ocurría en anteriores ocasiones.

Tabla 4. 22: Resultados del *Token Bucket. WRED* con *DiffServ*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	268	848	1116
Host B: 256 K	265	750	1015
Host A: 256 K	265	724	989
Host B: 512 K	524	596	1120
Host A: 256 K	263	650	913
Host B: 768 K	781	421	1202
Host A: 256 K	262	572	834
Host B: 1 M	1024	221	1245
Host A: 256 K	267	113	380
Host B: 1,768 M	1730	0	1730

En la tabla siguiente se observan los descartes realizados por *WRED* en la interfaz de salida del Router A. Se puede comprobar que sólo se realizan descartes mediante *WRED* y no por desbordamiento del buffer de la cola. También se puede observar que sólo se descartan paquetes marcados como OUT, independientemente de cual sea su procedencia. Estos resultados los muestra el *router* empleando el comando **show queueing interface serial 0/0**.

Tabla 4. 23: Paquetes descartados por *WRED* con *DiffServ*

Prueba	TRAFICO IN (paquetes)	TRAFICO OUT (paquetes)
1	0	133
2	0	106
3	0	115
4	0	115
5	0	67

Contrastando los datos anteriores con los obtenidos para las pruebas con *WRED* de un perfil único de descarte para todos los paquetes. Se puede ver que en el *WRED* con doble perfil no se descarta ningún paquete IN de la cola. Al descartar sólo paquetes OUT y al ser todo el tráfico *TCP*, se descartará un mayor número de paquetes de aquellas fuentes que generen más tráfico en exceso. Esto permite que cuando una fuente como el *host* B genera un porcentaje de tráfico IN mayor al 50% del enlace no se descarten sus paquetes, si no los del *host* A que genera más tráfico en exceso. De esta forma, *WRED* permite garantizar contratos mayores del 50% del enlace.

El problema vuelve a surgir cuando aparecen fuentes de tráfico no colaboradoras como las fuentes de tráfico *UDP*. Vemos los resultados para las pruebas llevadas a cabo usando una fuente generadora de tráfico *TCP* y otra *UDP*.

Tabla 4. 24: Resultados del *Token Bucket. WRED* con *DiffServ* tráfico *UDP y TCP*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	251	9219	9470	1.72
Host B: 256 K TCP	250	10	260	0.24
Host A: 256 K UDP	251	9219	9470	1.69
Host B: 1,768 M TCP	297	0	297	0.28

En la tabla anterior se han suprimido los resultados de varias pruebas debido a que no aportan novedades. En esos resultados se observa claramente que el contrato sólo se puede garantizar para el primer caso, que las dos fuentes tengan un contrato de 256 Kbps y además todo el ancho de banda en exceso se lo llevaría la fuente *UDP*. Para el último caso sólo se pueden garantizar 280 Kbps de los 1,768 Mbps que la fuente tiene contratados.

En la siguiente tabla se muestran los resultados de los descartes de paquetes realizados por *WRED*.

Tabla 4. 25: Paquetes descartados por *WRED* con *DiffServ* tráfico *UDP y TCP*

Prueba	TRÁFICO IN		TRÁFICO OUT	
	Random Drop	*Tail Drop*	Random Drop	*Tail Drop*
1	97/130656	17/20640	5839/6165984	106855/112851872
2	111/146336	19/24096	5527/5836512	107622/113648832

En la tabla se puede apreciar que se descartan paquetes IN y OUT ya sea mediante *WRED* o por desbordamiento de la cola. El tráfico *UDP* llena la cola de paquetes OUT haciendo que está crezca excesivamente provocando pérdidas de paquetes IN que no deberían ocurrir. Este problema se deriva de mezclar en una misma cola tráfico de los dos tipos: *TCP* y *UDP*.

Como conclusión, con la configuración de *WRED* con varios perfiles de descarte que se ha implementado aquí se pueden garantizar contratos de cualquier ancho de banda siempre que las fuentes generadoras del tráfico sean fuentes colaboradoras, es decir, *TCP*.

4.6.2 *CBWFQ*

Según lo explicado en el Capítulo 2, *WFQ* tiene dos beneficios principalmente:

- Proporciona la protección de cada clase de servicio asegurando un nivel mínimo del ancho de banda del puerto de salida independientemente del comportamiento de otras clases de servicio.

- Cuando se combina con acondicionadores de tráfico en las entradas de una red, *WFQ* garantiza un reparto equitativo del ancho de banda del puerto de salida de cada clase de servicio con un retardo limitado.

Una mejora a *WFQ* es *CBWFQ*. *CBWFQ* asigna paquetes a colas basándose en criterios de clasificación de paquetes definidos por el usuario. Por ejemplo, los paquetes se pueden asignar a

una cola en particular basándose en los bits del *IP DSCP*. Después de que los paquetes sean asignados a sus colas, podrán recibir un trato prioritario en función de los pesos configurados por el usuario.

Para estas pruebas se empleará la siguiente topología [47]:

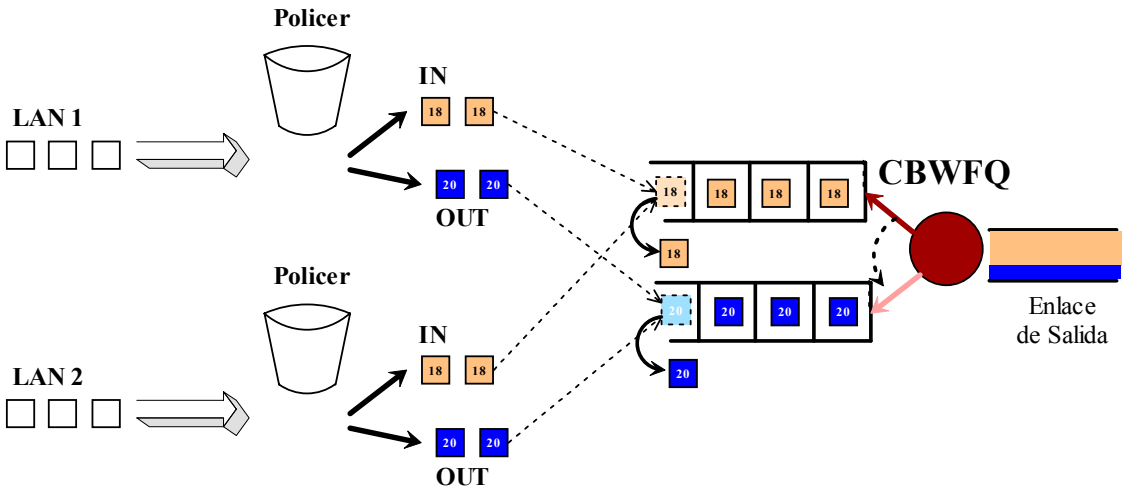

Figura 4. 13: Esquema del Funcionamiento de la Configuración de *CBWFQ*

En la figura anterior se muestra el proceso que lleva a cabo el Router A con los paquetes de las redes *LAN* 1 y 2 que le llegan. Primero, al igual que en casos anteriores, el tráfico se clasifica y se le aplica un *Token Bucket* que marca los paquetes como IN o como OUT. Después todos los paquetes IN de ambas redes se envían a una cola y todos los paquetes OUT se envían a otra [47]. Por último, *CBWFQ* se encargará de servir los paquetes de las colas. Los descartes de paquetes se realizarán cuando las colas se desborden.

El peso de cada una de las colas dependerá del contrato. Por ejemplo, si ambos contratos son de 256 Kbps, el total del tráfico IN será la suma de ambos, es decir, 512 Kbps (26% del enlace). El tráfico en exceso total será el resto del ancho de banda del enlace, esto es, 1,512 M (74% del enlace). Los porcentajes de *CBWFQ* para cada una de las pruebas son:

Tabla 4. 26: Porcentajes de *CBWFQ* para cada una de las pruebas

Pruebas	Cola Tráfico IN	Cola Tráfico OUT
1	26 %	74 %
2	38 %	62 %
3	50 %	50 %
4	63 %	37 %
5	100 %	0 %

4.6.2.1 Configuraciones

Las listas de acceso, las clases y las políticas de *policing* son las mismas que en caso general. En este caso se crearán dos clases más; una para clasificar el tráfico IN y otra para el tráfico OUT. Luego se creará una política que cree dos colas que se sirvan mediante *CBWFQ*. Como se ha expuesto a una de las colas irán a parar todos los paquetes IN y a la otra los OUT. En el *router* se haría lo siguiente:

Creación de las dos nuevas clases (además de las que había):

```
RouterA# configure terminal
RouterA(config)# class-map match-all traficoIN
RouterA(config-cmap)# match ip dscp 18
RouterA(config-cmap)# exit
RouterA(config)# class-map match-all traficoOUT
RouterA(config-cmap)# match ip dscp 20
RouterA(config-cmap)# exit
```

Creación de la nueva política (además de las que había):

```
RouterA# configure terminal
RouterA(config)# policy-map DiffServ
RouterA(config-pmap)# class traficoIN
RouterA(config-pmap-c)# bandwidth percent 26
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# class traficoOUT
RouterA(config-pmap-c)# bandwidth percent 74
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# exit
```

Se asocia la nueva política con la interfaz serial 0/0:

```
RouterA(config)# interface serial 0/0
RouterA(config-if)# service-policy output DiffServ
RouterA(config-if)# exit
```

Esta configuración es para el primero de los casos en el que los contratos de ambas redes eran iguales a 256 Kbps. Para cada una de las pruebas realizadas se deberá modificar estos valores según la tabla de porcentajes del apartado anterior.

4.6.2.2 Resultados y Comentarios

La siguiente tabla muestra los resultados obtenidos para cada una de las pruebas llevadas a cabo. En ella se puede apreciar que se cumplen perfectamente los contratos sea cual sea su tamaño. El problema sigue siendo el reparto del ancho de banda en exceso ya que continúa llevándose un porcentaje mayor la fuente cuyo contrato es menor.

Tabla 4. 27: Resultados del *Token Bucket. CBWFQ*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	261	1174	1435
Host B: 256 K	278	413	691
Host A: 256 K	266	846	1112
Host B: 512 K	539	432	971
Host A: 256 K	262	649	911
Host B: 768 K	804	362	1166
Host A: 256 K	261	734	995
Host B: 1 M	1025	42	1067
Host A: 256 K	268	94	363
Host B: 1,768 M	1751	0	1751

En el *router* no aparece que se lleven a cabo descartes cuando las fuentes de tráfico son *TCP* como en el caso anterior.

Para comprender que usando este método se logran garantizar de modo satisfactorio los contratos sea cual sea el tipo de tráfico, véanse los siguientes resultados en los que una de las redes *LAN* (*host* A) genera tráfico *UDP* y la otra (*host* B) *TCP*:

Tabla 4. 28: Resultados del *Token Bucket*. *CBWFQ* con tráfico *UDP* y *TCP*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	251	9219	9470	1,7
Host B: 256 K TCP	260	30	290	0,25
Host A: 256 K UDP	251	9219	9470	1,46
Host B: 512 K TCP	516	4	520	0,49
Host A: 256 K UDP	251	9219	9470	1,23
Host B: 768 K TCP	763	2	765	0,73
Host A: 256 K UDP	251	9219	9470	0,98
Host B: 1 M TCP	1007	3	1010	0,97
Host A: 256 K UDP	251	9219	9470	0,28
Host B: 1,768 M TCP	1739	0	1739	1,67

Se observa que de todas las configuraciones estudiadas, es la única que logra garantizar los contratos sea cual sea el tamaño y el tipo de tráfico. Esto se debe a que *CBWFQ* separa el tráfico en exceso en una cola y el tráfico IN en otra. Así, como la mayor parte del tráfico *UDP* es marcado como OUT, se aislará casi todo en una de las colas y no influirá en el tráfico IN de la otra. Esto permite garantizar los contratos, ya que el tráfico IN, independientemente de que sea *UDP* o *TCP*, no sobrepasará el ancho de banda del enlace.

En la siguiente tabla se muestran los descartes llevados a cabo por desbordamiento de las colas:

Tabla 4. 29: Paquetes descartados en *CBWFQ* con tráfico *UDP y TCP*

Prueba	Paquetes IN descartados	Paquetes OUT descartados
1	0	113186
2	0	116476
3	7	119926
4	0	123550
5	9	133810

Se aprecia claramente que prácticamente todos los descartes se producen por desbordamiento de la cola de tráfico OUT. Por esta razón, los flujos de tráfico *UDP* no impiden que se garanticen los contratos.

El problema sigue siendo el reparto equitativo del ancho de banda en exceso. Sobre todo cuando hay tráfico *UDP*, se observa que las fuentes de tráfico *UDP* se quedan con todo el ancho de banda en exceso utilizando esta configuración.

4.6.3 *CBWFQ* con *WRED*

Ya se han llevado a cabo configuraciones con *CBWFQ* y con *WRED* por separado y se han obtenido y comentado los resultados de sus pruebas correspondientes. En las siguientes pruebas se activarán ambas herramientas a la vez. Con esta configuración además de lograr garantizar los contratos, se solucionará el problema del reparto equitativo del ancho de banda en exceso que no se conseguía con ninguna de las otras configuraciones.

Primero, se clasificarán los paquetes separándolos por red de origen. Luego, se procesarán mediante un *Token Bucket* para ver si cumplen o no los contratos y serán marcados. Esta vez se utilizarán cuatro valores de *DSCP* diferentes, dos para cada red *LAN*. Así, los paquetes de la red *LAN* 1 que cumplan el contrato se marcarán con un valor de *DSCP* de 18 y los que lo sobrepasen con un *DSCP* de 20. Los paquetes IN de la red *LAN* 2 se marcarán con un *DSCP* de 10 y los OUT con un *DSCP* de 12.

Una vez marcados los paquetes, se enviarán a dos colas diferentes: una para los paquetes IN y OUT de la red *LAN* 1 y otra para los paquetes IN y OUT de la red *LAN* 2. Cada una de esas colas usará un *WRED* con dos perfiles de descarte diferentes: uno para el tráfico IN y otro para el tráfico OUT. Ambas colas se servirán usando *CBWFQ*.

La siguiente figura muestra un esquema de lo expuesto:

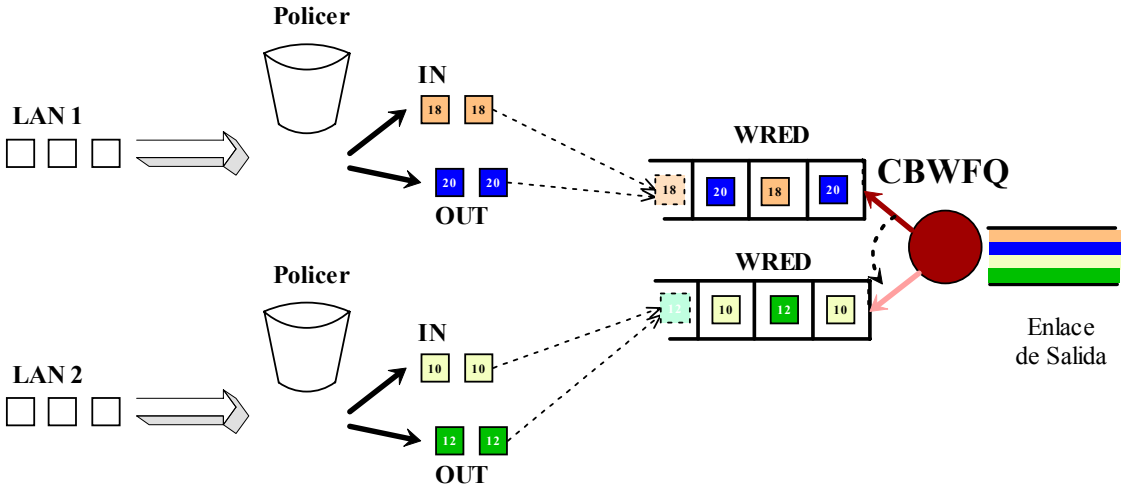

Figura 4. 14: Esquema del Funcionamiento de la Configuración de *CBWFQ* con *WRED*

Los parámetros de *WRED* para ambas colas serán los mismos que cuando se usó *WRED* con doble perfil de descarte, es decir, para los paquetes que los *Token Bucket* marquen como IN los parámetros de *WRED* serán: un umbral mínimo de 40, un umbral máximo de 70 y un *mark probability denominador* de 50.Y para los paquetes marcados como OUT por el *Token Bucket* los parámetros de *WRED* serán: un umbral mínimo de 10, un umbral máximo de 40 y un *mark probability denominador* de 5.

Los parámetros de configuración de *CBWFQ*, esto es, el peso de cada una de las colas, dependerán del contrato y se establecerán de modo que se intente repartir de forma equitativa el ancho de banda en exceso, para solucionar los problemas de las anteriores configuraciones. Por ejemplo: si una red *LAN* tiene contratados 256 Kbps y la otra 768 Kbps, el ancho de banda total contratado será de 1 Mbps al igual que el ancho de banda en exceso. Si se reparte ese ancho de banda sobrante al 50% entre ambas redes cada una tendrá derecho a 512 Kbps. Por tanto para la red *LAN* cuyo contrato era de 256 Kbps ocupará 256 + 512 Kbps del enlace de salida, es decir, un 37% del ancho de banda del enlace. La red *LAN* de contrato 768 Kbps ocupará 768 + 512 Kbps del enlace de salida, un 63% del ancho de banda del enlace.

Los parámetros para cada una de las configuraciones son:

Tabla 4. 30: Pesos de las colas para los diferentes contratos. *CBWFQ* con *WRED*

Pruebas	Cola Tráfico *LAN* 1	Cola Tráfico *LAN* 2
1	50 %	50 %
2	44 %	56 %
3	38 %	62 %
4	32 %	68 %
5	12 %	88 %

4.6.3.1 Configuraciones

En este caso, de la configuración realizada para el resto de pruebas sólo sirven la creación de las listas de control de acceso y sus clases correspondientes. Por lo tanto, en esta ocasión se crearán dos nuevas clases; una para el tráfico marcado con los *DSCPs* 18 y 20 y otra para el tráfico marcado con los *DSCPs* 10 y 12. También se creará una nueva política para configurar *CBWFQ* con *WRED*. Y uno de los *Token Bucket* creados para el resto de configuraciones será configurado para marcar los paquetes de una de las redes con los valores de *DSCP* 10 y 12 (recuérdese que antes ambos *Token Bucket* marcaban todos los paquetes con dos *DSCPs*, el 18 y el 20). La configuración en el *router* será:

Creación de las dos nuevas clases (además de las clases trafico3.14-2.5 y trafico1.5-2.5):

```
RouterA# configure terminal
RouterA(config)# class-map match-all traficoDSCP18_20
RouterA(config-cmap)# match ip dscp 18 20
RouterA(config-cmap)# exit
RouterA(config)# class-map match-all traficoDSCP10_12
RouterA(config-cmap)# match ip dscp 10 12
RouterA(config-cmap)# exit
```

Con estas clases se clasifican los paquetes a la entrada de la interfaz serial después de que hayan sido marcados por los *Token Buckets*.

Como se ha dicho, los *traffic policing* a la entrada del *router* serán iguales a los de las anteriores configuraciones salvo que ahora los valores de los *DSCPs* de cada una de las redes serán diferentes. Como ejemplo:

```
RouterA# configure terminal
RouterA(config)# policy-map TrafficPolicing3.14_2.5
RouterA(config-pmap)# class trafico3.14-2.5
RouterA(config-pmap-c)# police 256000 48000 96000 conform-action
set-dscp-transmit 18 exceed-action set-dscp-transmit 20
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# exit
RouterA(config)# policy-map TrafficPolicing1.5_2.5
RouterA(config-pmap)# class trafico1.5-2.5
RouterA(config-pmap-c)# police 256000 48000 96000 conform-action
set-dscp-transmit 10 exceed-action set-dscp-transmit 12
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# exit
```

Creación de la política de Servicios Diferenciados que emplea *CBWFQ* con *WRED*:

```
RouterA# configure terminal
RouterA(config)# policy-map DiffServ
RouterA(config-pmap)# class traficoDSCP18_20
RouterA(config-pmap-c)# bandwidth percent 50
RouterA(config-pmap-c)# random-detect dscp-based
RouterA(config-pmap-c)# random-detect dscp 18   40    70    50
RouterA(config-pmap-c)# random-detect dscp 20   10    40    5
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# class traficoDSCP10_12
RouterA(config-pmap-c)# bandwidth percent 50
RouterA(config-pmap-c)# random-detect dscp-based
RouterA(config-pmap-c)# random-detect dscp 10   40    70    50
RouterA(config-pmap-c)# random-detect dscp 12   10    40    5
RouterA(config-pmap-c)# exit
RouterA(config-pmap)# exit
```

Mediante esta configuración se han creado dos colas; una para los paquetes marcados con los *DSCPs* 18 y 20 y otra para los paquetes marcados con los *DSCPs* 10 y 12. A cada una de esas colas se le asigna la mitad del ancho de banda del enlace de salida. Además, se configura *WRED* dentro de cada una de las colas para que se encargue del descarte de paquetes, descartando antes los paquetes marcados como OUT que los paquetes IN.

Se asocia la política con la interfaz de salida serial 0/0:

```
RouterA(config)# interface serial 0/0
RouterA(config-if)# service-policy output DiffServ
RouterA(config-if)# exit
```

4.6.3.2 Resultados y Comentarios:

En la tabla siguiente se pueden ver los resultados de las pruebas llevadas a cabo con la anterior configuración. En ella se observa que se cumplen los contratos sea cual sea su tamaño, aunque estos superen el 50% del ancho de banda del enlace. Por otro lado, se ha conseguido repartir el ancho de banda en exceso a partes iguales entre ambas redes, cosa que no se había conseguido con ninguna de las configuraciones anteriores.

Tabla 4. 31: Resultados del *Token Bucket. CBWFQ* con *WRED*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)
Host A: 256 K	261	771	1032
Host B: 256 K	264	782	1046
Host A: 256 K	262	653	915
Host B: 512 K	523	643	1166
Host A: 256 K	262	539	801
Host B: 768 K	781	500	1281
Host A: 256 K	268	437	702
Host B: 1 M	1041	359	1400
Host A: 256 K	264	56	320
Host B: 1,768 M	1778	0	1778

En la siguiente tabla se muestran los descartes llevados a cabo por *WRED* para cada una de las colas. Estos valores se obtienen usando el comando **show policy-map interface serial 0/0.** En ella se puede apreciar que casi nunca se descartan paquetes IN, solamente en el último caso y muy pocos.

Tabla 4. 32: Paquetes descartados por *WRED*. *CBWFQ* + *WRED*

Prueba	*DSCP*	Paquetes descartados por *WRED*
1	18	0
	20	72
	10	0
	12	69
2	18	0
	20	62
	10	0
	12	77
3	18	0
	20	58
	10	0
	12	78
4	18	0
	20	51
	10	0
	12	87
5	18	2
	20	18
	10	9
	12	0

Para ver de una manera más clara los resultados anteriores, véase en que número llegan los paquetes al Router B, esto es, después de realizar todos los descartes:

Tabla 4. 33: Paquetes que llegan al Router B

Prueba	Tráfico *DSCP* 18 (Kbps)	Tráfico *DSCP* 20 (Kbps)	Tráfico *DSCP* 10 (Kbps)	Tráfico *DSCP* 12 (Kbps)	Tráfico Total (Kbps)
1	261	764	264	782	2071
2	262	647	523	636	2067
3	262	534	781	492	2068
4	265	432	1041	350	2088
5	264	54	1777	0	2095

En anteriores configuraciones, lo que parecía funcionar correctamente cuando el tráfico a tratar era tráfico *TCP*, no servía cuando alguna de las fuentes generaba tráfico *UDP*. Por tanto, los siguientes resultados corresponden a las pruebas llevadas a cabo utilizando fuentes de tráfico *UDP*.

Tabla 4. 34: Resultados del Token Bucket. *CBWFQ* + *WRED* con tráfico *UDP* y *TCP*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	251	9219	9470	0,98
Host B: 256 K TCP	260	752	1012	0,96
Host A: 256 K UDP	251	9219	9470	0,86
Host B: 512 K TCP	516	624	1140	1,09
Host A: 256 K UDP	251	9219	9470	0,74
Host B: 768 K TCP	773	485	1258	1,2

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	251	9219	9470	0,63
Host B: 1 M TCP	1030	347	1377	1,32
Host A: 256 K UDP	251	9219	9470	0,24
Host B: 1,768 M TCP	1767	0	1767	1,7

En la tabla anterior se muestran los resultados del marcado de paquetes del *Token Bucket*, pero no se puede apreciar el número de paquetes de cada *DSCP* que llegan a su destino. Para ver el número de paquetes que llega a su destino habrá que descontarle a los anteriores resultados los descartes realizados en la interfaz serial 0/0. Así, los paquetes que llegarán a su destino en realidad serán:

Tabla 4. 35: Caudal de Tráfico que Llega a su Destino Después de Realizar los Descartes. *CBWFQ + WRED* con tráfico *UDP y TCP*

Fuente y Contrato	Tráfico IN (Kbits/s)	Tráfico OUT (Kbits/s)	Tráfico Total (Kbits/s)	Tráfico Total que Llega (Mbits/seg)
Host A: 256 K UDP	246	743	989	0,98
Host B: 256 K TCP	259	745	1004	0,96
Host A: 256 K UDP	245	615	860	0,86
Host B: 512 K TCP	516	618	1134	1,09
Host A: 256 K UDP	246	500	746	0,74
Host B: 768 K TCP	773	479	1252	1,2
Host A: 256 K UDP	245	385	630	0,63
Host B: 1 M TCP	1030	340	1370	1,32
Host A: 256 K UDP	236	9	245	0,24
Host B: 1,768 M TCP	1766	0	1766	1,7

El caudal total del tráfico que llega de cada red *LAN* a su destino viene reflejado en la última columna. Como se puede comprobar por los resultados obtenidos en todas las pruebas realizadas, con este tipo de configuración se logran garantizar los contratos sea cual sea su tamaño e independientemente del tráfico que se genere. Además, se consigue un reparto del ancho de banda en exceso al 50% para ambas redes.

Todo esto se logra mediante la separación de los tráficos de cada red en una cola diferente y la asignación de los pesos adecuados a cada cola. En el caso en el que se usaba *CBWFQ* con dos colas una para el tráfico IN y otra para el tráfico OUT se podían garantizar los contratos incluso con fuentes *UDP* ya que la mayor parte del tráfico *UDP* era marcado como OUT y enviado a una cola aislada, por lo tanto, no influía negativamente en el tráfico IN. Pero el problema estaba en la imposibilidad de repartir por igual el ancho de banda en exceso entre ambas redes, debido a que la red generadora de tráfico *UDP* consumía todos los recursos de la cola del tráfico OUT. Separando el tráfico de las redes en colas separadas, el mal comportamiento del tráfico de una red no afectará al contrato de la otra. Usando *WRED* con dos perfiles de descarte distintos, se les dará preferencia a los paquetes IN frente a los OUT dentro de una misma cola, es decir, los paquetes que no cumplan el contrato tendrán más probabilidad de ser descartados que los paquetes que lo cumplan para el tráfico de una red.

4.7 Conclusiones

Se han visto distintas posibilidades de configuración en los *routers* para intentar garantizar contratos a la vez que se intentaba repartir por igual el ancho de banda en exceso y se han llevado a cabo pruebas con diferentes tipos de tráfico *TCP* y *UDP* para ver como reaccionaban los *routers* ante diversas situaciones. A continuación se puede ver una tabla con un resumen de los resultados obtenidos:

Tabla 4. 36: Resumen de los Resultados Obtenidos

		TCP		***UDP + TCP***	
	Configuración	**Garantizar Contratos**	**Reparto equitativo del ancho de banda en exceso.**	**Garantizar Contratos**	**Reparto equitativo del ancho de banda en exceso.**
Sin *DiffServ*	*FIFO* con *Tail Drop*	Sí, pero cuando ninguno de los contratos sobrepase el 50% del ancho de banda del enlace.	No	No	No
	FIFO con *WRED*	Sí, pero cuando ninguno de los contratos sobrepase el 50% del ancho de banda del enlace.	No	No	No
	WFQ	Sí, pero cuando ninguno de los contratos sobrepase el 50% del ancho de banda del enlace.	No	Sí, pero cuando ninguno de los contratos sobrepase el 50% del ancho de banda del enlace.	No
Con *DiffServ*	*WRED* con doble perfil de Descarte	Sí, sea cual sea el contrato.	No	No	No
	CBWFQ	Sí, sea cual sea el contrato.	No	Sí	No
	CBWFQ con *WRED*	Sí, sea cual sea el contrato.	Sí	Sí	Sí

Por lo tanto, se puede ver que solamente con la última configuración llevada a cabo es posible asegurar los contratos sea cual sea su tamaño y para cualquier tipo de tráfico. Además de repartir de modo equitativo el ancho de banda en exceso.

Pero también hay que notar que conforme se han ido haciendo más precisas las configuraciones, se ha ido aumentando en complejidad y se ha ido disminuyendo en escalabilidad. Por ejemplo, las configuraciones sin Servicios Diferenciados no dependen del contrato por lo tanto su implementación es mucho más sencilla y su escalabilidad muy alta. Por otro lado, dentro de las configuraciones con Servicios Diferenciados, *CBWFQ* con *WRED* será más complicado de configurar y menos escalable que *WRED* con doble perfil de descarte, ya que en el primero se necesitan hacer cálculos específicos de los pesos de las colas según los contratos.

Capítulo 5
Conclusiones

5.1 Conclusiones

Los diferentes tipos de tráfico emergentes actualmente en las redes *IP* tienen unas necesidades de retardo, varianza del retardo, caudal y pérdidas muy diversas y requieren un tratamiento individualizado. Se ha presentado la arquitectura de los Servicios Diferenciados como una muy buena solución para solventar esas necesidades. En esta presentación se han definido los distintos elementos que la componen, como son:

- El dominio de Servicios Diferenciados en el que todos los nodos son capaces de proporcionar un tratamiento específico a los flujos de tráfico en función del valor del campo *DSCP* de la cabecera *IP* de cada paquete. Este dominio está compuesto de dos tipos de nodos: los nodos frontera que llevan a cabo funciones de clasificación y acondicionamiento del tráfico entrante o saliente del dominio y los nodos interiores que le dan un tratamiento particular a cada paquete dependiendo del valor de *DSCP* de su cabecera *IP*. Existen dos tipos de clasificadores: los que clasifican basándose solamente en el valor del *DSCP* (BA classifiers) y los que clasifican basándose en el valor de la combinación de uno o más campos de la cabecera (Clasificadores Multi-Campo). Un acondicionador del tráfico puede contener los siguientes elementos: un medidor, un marcador, un espaciador o un descartador de tráfico.

- También se vio lo que es un *PHB* que es el tratamiento particular que un nodo realiza sobre un flujo de tráfico con unas características determinadas, es decir, las funciones de programación, encolado, espaciado, etc que un nodo le aplica a los paquetes de ese flujo. Se vió que hay varios *PHB*s definidos por el *IETF* que tienen asignados ciertos valores de *DSCP*, como son: el *PHB* por defecto que significa tratamiento de Best Effort y tiene asociado el valor de *DSCP* '000000', el *PHB* selector de clase diseñado para preservar la compatibilidad con el anterior *IP* Precedence y tiene asociados un *DSCP* de la forma 'xxx000', el *Expedited Forwarding PHB* que significa proporcionar un servicio de pocas pérdidas, bajo retardo, pequeña varianza del retardo y asegurar un ancho de banda constante y el valor de *DSCP* asociado con este *PHB* es '101110' y finalmente el *Assured Forwarding PHB* por medio del cuál a los paquetes de un determinado flujo de tráfico se les reserva cierta cantidad del ancho de banda de la interfaz de salida y pueden ser descartados con diferente probabilidad.

- A continuación, se explicó en detalle las diferentes técnicas de servicio y gestión de colas. Entre las disciplinas para servir colas están:

  - *FIFO* que es la más básica y en la que todos los paquetes se tratan igual, el primero que llega es el primero en salir,.
  - PQ que está diseñado para proporcionar diferenciación de servicios, este tipo de disciplinas cuanta con varias colas y las sirve según su prioridad.
  - FQ diseñado para que cada flujo tenga un acceso justo a los recursos de la red evitando que un flujo de ráfagas consuma más ancho de banda que la parte que le corresponde.
  - *WFQ* que viene a solucionar las limitaciones de FQ, así *WFQ* soporta flujos con diferentes requerimientos de ancho de banda y soporta paquetes de longitud variable de modo que flujos con paquetes pequeños no salgan perjudicados respecto a flujos con paquetes grandes.

- *CBWFQ* asigna paquetes a las colas basándose en criterios de clasificación de paquetes definidos por el usuario, por ejemplo los paquetes se pueden asignar a una cola basándose en el valor del campo *DSCP*.

- Finalmente, antes de explicar los mecanismos para evitar la congestión se vio Tail Drop que significaba la ausencia total de un gestor de la memoria de la cola, luego se explicó RED que lleva a cabo un descarte aleatorio de paquetes de la cola antes de que la congestión se convierta en un problema y *WRED* que es una extensión de *RED* que permite asignar diferentes perfiles de descarte *RED* a diferentes tipos de tráfico proporcionando una precisión mayor de control que el *RED* clásico. Así, los beneficios de la gestión activa de las colas comparadas con *Tail Drop* incluyen: La eliminación de la sincronización global de fuentes *TCP* que da como resultado un uso más eficiente del ancho de banda de la red, el soporte de fluctuaciones momentáneas en el tamaño de la cola, que permiten absorber ráfagas sin descartar paquetes y causar que los *host* reduzcan sus caudales cuando reducen sus tasas de transmisión y la habilidad para controlar el tamaño de la cola influyendo en la media del retardo de encolamiento a través del *router*.

Debido a las diferentes necesidades de los tipos de tráfico actuales los vendedores de *routers* deben utilizar las diferentes técnicas explicadas o una combinación de ellas para intentar evitar y gestionar, cuando ocurra, la congestión. Así, como aparece en la implementación de los Servicios Diferenciados que realiza Cisco *Systems* del capítulo 3, se emplean una combinación de estas disciplinas para solucionar los problemas de la congestión. Por ejemplo, Cisco *IOS* emplea una combinación de *WRED* y *CBWFQ* a la hora de servir las colas del *Assured Forwarding PHB*.

En el Capítulo 3 se ha hecho un estudio de las herramientas que emplea el software de Cisco *IOS* para implementar la arquitectura de Servicios Diferenciados. Para ello, primero se ha realizado una introducción al software de red que llevan instalados los routers de Cisco explicando el modo de usarlo y la forma de actualizarlo ya que para la realización de las pruebas se necesita una versión reciente de ese software.

Luego se han analizado las herramientas de que dispone el software de Cisco para implementar cada uno de los elementos de los Servicios Diferenciados y como configurarlas.

- Para las tareas de clasificación de los paquetes el software Cisco *IOS* utiliza los **class-maps** dentro del *MQC* con los que se definen las clases de tráfico, también emplea las listas de acceso para llevar a cabo un filtrado selectivo de los flujos de tráfico y poder así diferenciarlos. Cabe resaltar que los acondicionadores del tráfico como el *traffic policing* también pueden llevar a cabo tareas de clasificación cuando marcan los paquetes de modo diferente dependiendo de si estos cumplen o no la tasa contratada.

- Para el acondicionamiento del tráfico Cisco emplea *Class-Based Policing* y *Class-Based Shaping*. El primero de ellos permite limitar la tasa de transmisión de una clase de tráfico y marcar los paquetes con valores apropiados de *DSCP*. Con el segundo, *Class-Based Shaping*, permite controlar el tráfico que abandona una interfaz para casar su tasa de transmisión con la velocidad de la interfaz remota. Para el marcado de los paquetes Cisco dispone además de los acondicionadores del tráfico nombrados de la herramienta *Class-Based Packet Marking* que lo que hace es marcar los paquetes de una determinada clase definida previamente mediante el comando **class-map**.

- El tratamiento particular (PHB) que un nodo aplique a los paquetes de un determinado flujo de tráfico (BA) como se vio, depende en gran medida de las técnicas de encolamiento y de cómo el nodo gestione y controle la congestión de los enlaces. El software de Cisco *IOS* emplea *CBWFQ* y *LLQ* para servir las colas en periodos de

congestión. Mediante *CBWFQ* se pueden asignar diferentes porciones del ancho de banda del enlace de salida a cada una de las clases definidas y se emplea para proporcionar el *Assured Forwarding PHB*. *LLQ* permite desencolar paquetes con necesidades de bajo retardo y pequeña varianza del retardo antes que paquetes de otras clases implementando así el *Expedited Forwarding PHB*. Para evitar que se produzca la congestión Cisco dispone de *WRED* con el que realiza descartes antes de que la congestión se convierta en un problema. *WRED* podrá usar los valores de *DSCP* de la cabecera de los paquetes a la hora de descartarlos y así implementar la parte del *Assured Forwarding PHB* correspondiente al descarte de paquetes.

Para configurar de un modo sencillo mediante una interfaz de comandos todas estas características de *DiffServ* el software de Cisco *IOS* proporciona el *MQC*. Con él simplemente harán falta tres pasos a la hora de configurar todas las herramientas de Servicios Diferenciados que ofrece Cisco estos pasos son: primero definir la clase de tráfico con el comando **class-map**, segundo asociar esa clase con una o más políticas de Calidad de Servicio (funciones de policía, espaciado, encolamiento, etc) creando una *service police* y por último asignar esa service police a una interfaz determinada. El capítulo finaliza explicando *CEF* que es una avanzada tecnología de conmutación de la capa *IP*que debe ser activada para poder configurar algunas de las herramientas anteriormente nombradas.

Finalmente en el capítulo 4 el objetivo era el de comprobar el funcionamiento de los *routers* Cisco de la serie 2600 en un entorno de Servicios Diferenciados con la intención de ver si los proveedores de servicios de Internet podían garantizar contratos empleando los mecanismos de diferenciación de servicios de ese tipo de routers y conseguir al mismo tiempo repartir el ancho de banda en exceso de la manera más justa posible. Para ello se llevaron a cabo una serie de experimentos sobre una topología de red en la que se creaba un cuello de botella entre dos routers de Cisco y se generaba tráfico desde dos supuestas redes *LAN*. Usando diferentes configuraciones de Servicios Diferenciados en uno de los routers, se obtuvieron resultados sobre como era tratado ese tráfico. Lo primero que se hacía en todas estas pruebas era medir el tráfico a la entrada y marcarlo usando la herramienta *Class-Based Policing* dependiendo de si éste sobrepasaba o no el contrato establecido. Si el tráfico estaba dentro del contrato se marcaba con un *DSCP* y si lo sobrepasaba se marcaba con otro pero usando esta herramienta no se descartaba ningún paquete. Se obtuvieron resultados para dos grupos de pruebas: configuraciones sin ningún tipo de diferenciación de servicios y configuraciones con diferenciación de servicios. Para el primer grupo de pruebas todos los paquetes se trataban por igual independientemente de cómo estuvieran marcados, es decir, independientemente de si pertenecían a una red *LAN* o a otra o sí cumplían o no lo contratos. Los resultados obtenidos indicaron que los contratos sólo se podían garantizar cuando estos no superasen el 50% del ancho de banda del enlace congestionado, que en ninguno de los casos se repartía de forma equitativa el ancho de banda en exceso y que en ninguno de los casos, salvo con *WFQ*, se podían garantizar los contratos si alguna de las fuentes generaba tráfico no 'colaborador' como *UDP*.

Para el segundo grupo de pruebas en el que se aplicaban las características de los Servicios Diferenciados se constató que en todos los casos era posible garantizar los contratos para tráfico *TCP* y *UDP* incluso cuando estos superasen el 50% del ancho de banda del enlace, excepto cuando se empleaba *WRED* con doble perfil de descarte para el que no era posible garantizar los contratos cuando coexistían tráfico *UDP* y *TCP*. Por otro lado se vio que solamente se podía lograr un reparto justo del ancho de banda en exceso cuando se empleara *CBWFQ* con *WRED*. Una tabla resumen de los resultados obtenidos se puede ver en el apartado de "conclusiones" del Capítulo 4.

Con todo esto se ve que conforme se obtuvieron mejores resultados la complejidad de las configuraciones aumentó y la escalabilidad disminuyó. Por todo ello se concluye que existe la posibilidad de que los proveedores de servicios de Internet garanticen contratos de una forma justa usando elementos de la arquitectura de Servicios Diferenciados cuando el número de redes

o contratos no es excesivamente grande pero están más indicados a la hora de distinguir el tráfico de las diferentes aplicaciones que a la hora de distinguir el tráfico de redes particulares. Los problemas que presenta el emplear los elementos de la arquitectura de Servicios Diferenciados a la hora de garantizar contratos son:

- El número de contratos diferentes no debe de ser excesivamente grande debido a que el número de DSCPs con los que se diferencian los paquetes de cada contrato son limitados.

- La configuración de estos contratos es relativamente compleja y además se debe implementar en todos los nodos del dominio de Servicios Diferenciados si se sigue la arquitectura definida por el *IETF* suponiendo un gran esfuerzo. Por ejemplo, en el caso de CBWFQ y CBWFQ con WRED se tiene que el peso de cada una de las colas depende del contrato y si los contratos varían o se añade alguno, esos pesos se tienen que volver a calcular.

- La dificultad de crecimiento debido a que cuando aparece un nuevo contrato se deben modificar todas las políticas de todos los nodos creadas, para adaptarse al nuevo reparto de ancho de banda del canal.

Por todo ello, podría resultar más efectivo utilizar los Servicios Diferenciados para dar un tratamiento diferente a los flujos de tráfico generados por las distintas aplicaciones ya que así:

- El número de protocolos existente no es excesivamente grande y se pueden asociar con los DSCPs actuales.

- La configuración de las herramientas para dar un tratamiento particular a cada tipo de tráfico está más estudiada, controlada y estandarizada de modo que una vez aplicado en los nodos del dominio no es necesario cambiarla continuamente.

- Mantiene una relativa independencia del crecimiento de las redes manteniéndose estable sin la necesidad de actualización, ya que aunque aumente el número de fuentes los protocolos serán los mismos y los nodos los identificarán.

Así, la configuración y prueba de los routers de Cisco para diferenciar los flujos de tráfico por la aplicación que los genera puede ser una línea futura de trabajo muy interesante.

Anexo A
Listas de Control de Acceso

A.1 Listas de Control de Accesso [48]

Las listas de control de acceso (ACL) son listas de instrucciones que se aplican a la interfaz del router y definen cómo los paquetes se introducen en las interfaces de entrada, se distribuyen por el router y abandonan las interfaces de salida del mismo. Estas listas las emplea el router para saber qué paquetes debe aceptar y cuales denegar.

Las ACL ofrecen la posibilidad de gestionar el tráfico permitiendo o prohibiendo a los flujos de tráfico entrar o salir del router. Las ACL se pueden configurar en el router para controlar el acceso a una red o subred. Por ejemplo, en la UPCT, las ACL se podrían usar para evitar que el tráfico de los estudiantes accediera a la red administrativa.

Las ACL nos van a permitir filtrar el tráfico que entra a una interfaz para poder clasificarlo en diferentes clases de tráfico y una vez clasificado aplicarle diferentes políticas de QoS, como pueden ser funciones policía (policing), espaciado del tráfico (shaping), marcado de los paquetes (marking), etc. Por ejemplo, mediante ACL se podría filtrar todo el tráfico VoIP, clasificarlo en una clase dentro del router y posteriormente darle preferencia, mediante políticas de QoS, a ese tipo de tráfico frente a otros.

Otras funciones para las que las ACL se pueden emplear son: limitar el tráfico de la red e incrementar el rendimiento de la misma, proporcionar un nivel de seguridad básico, decidir qué tipos de tráfico se reenvían o se bloquean en las interfaces del router, etc.

Hay que tener en cuenta el orden de colocación de las instrucciones ACL ya que el software de Cisco IOS comprueba el paquete con cada instrucción condicional en el orden en el que las instrucciones fueron creadas y cuando encuentra una coincidencia ya no se comprueban más instrucciones condicionales. Así, si se crea una instrucción que permita todo el tráfico al principio, no se comprobarán las instrucciones que hayan sido creadas después.

Cuando llega un paquete al router lo primero que hace éste es comprobar si el paquete se puede enrutar, es decir, si hay alguna entrada en la tabla de encaminamiento para el destino de ese paquete (esto lo hace el router existan o no ACL). Si el paquete se puede enrutar, el router comprobará entonces si la interfaz de entrada tiene una ACL. En el caso de que exista dicha ACL, el paquete se probará con las condiciones de la lista. Si se permite el paquete, éste es cotejado con las entradas de la tabla de enrutamiento con el fin de determinar que las *ACL* de la interfaz de destino no filtran los paquetes que se originen en el propio *router*, sino los paquetes de otros orígenes. Lo siguiente que hará el router será comprobar si la interfaz de destino tiene una ACL. Si no la tiene, el paquete se enviará directamente a la interfaz de destino.

Los pasos para crear una ACL serán los siguientes:

Paso 1: Definir la ACL empleando el comando **access-list** seguido de: el número-de-lista-de-acceso que asigna un número a la ACL según el protocolo e identifica a cada ACL de forma unívoca, las sentencias **permit** o **deny** que indican si se permitirá o se denegará el acceso del paquete que cumpla las condiciones de la ACL, las condiciones que se deben comprobar para el paquete. La sentencia sería la siguiente:

```
RouterA(config)#access-list número-de-lista-de-acceso {permit | deny}
{probar-condiciones}
```

Las condiciones a comprobar pueden ser diversas: dirección origen, dirección destino, protocolos de transporte, números de puerto, etc o una combinación de éstas. Las condiciones dependen del número de lista de acceso que se haya elegido. Si el número está entre 1 y 99 se estarán creando las llamadas ACL estándar. Las *ACL* estándar se usan cuando se desea bloquear todo el tráfico de una red o de un *host* específico, permitir todo el tráfico de una red específica o denegar paquetes de protocolos. Así, la sintaxis completa del comando de creación de una ACL estándar sería:

```
Router(config)#access-list número-de-lista-de-acceso {permit | deny} source
(wildcard-de-origen] [log]

Donde:
```

Tabla A. 1: Parámetros de las ACLs estándard

Parámetro	Descripción
Número-de-lista-de-acceso	El número de una *ACL*. Es un número decimal del 1 al 99 (en una *ACL IP* estándar).
deny	Prohibe (deniega) el acceso si no coinciden las condiciones.
permit	Autoriza (permite) el acceso si coinciden las condiciones.
origen	El número de la red o *host* desde el que se está enviando el paquete. Hay dos formas de especificar el origen: • Utilizando una cantidad de 32 bits en un formato decimal con puntos. • Utilizando la palabra clave any como abreviatura de un origen y un *wildcard*-de-origen de 0.0.0.0 255.255.255.255.
wildcard-de-origen	(opcional) Los bits de *wildcard* que hay que aplicar al origen. Existen dos maneras de especificar el *wildcard*-de-origen: • Utilizando una cantidad de 32 bits en formato decimal con puntos. Colocando unos en las posiciones de bit que se desea ignorar. • Utilizando la palabra clave any como abreviatura de un origen y un *wildcard*-de-origen de 0.0.0.0 255.255.255.255.
log	(Opcional) Hace que se envíe a la consola un mensaje de registro informativo acerca del paquete que coincida con la entrada (el nivel de mensajes que se registra en la consola lo controla el comando logging console). El mensaje incluye el número *ACL*, si el paquete ha sido permitido o denegado, la dirección de origen y el número de paquetes. El mensaje se genera en el primer paquete que coincida a intervalos de cinco minutos, incluyendo el número de paquetes permitidos o denegados en el intervalo previo de cinco minutos.

Ejemplo:

```
access-list 28 permit 192.168.12.0 0.0.0.255
access-list 28 permit 215.33.0.0 0.0.255.255
```

(Nota: todos los demás accesos quedan negados implícitamente)

En el siguiente ejemplo la ACL estándar permite el acceso de los host de las redes 192.168.12.0 y 215.33.0.0. Todo *host* cuya dirección no pertenezca a alguna de esas dos redes será rechazado.

Por otro lado, si el número de la lista de acceso está en el intervalo de 100 a 199 se estará creando una ACL llamada ACL extendida. Las *ACL* extendidas comprueban tanto las

direcciones de origen como de destino de los paquetes. También comprueban si hay protocolos específicos, números de puerto y otros parámetros. Por lo tanto, proporcionan un nivel de control superior que las *ACL* estándar y serán las que utilicemos a la hora de configurar los routers en las pruebas experimentales que se realizan en el capítulo 4.

La forma completa del comando para una ACL extendida es:

```
Router(config-if)#access-list número-de-lista-de-acceso {permit | deny }
protocol source [ máscara-de-origen destination máscara-de-destino
[ operator operando ] [established ] [log ]
```

Donde los parámetros tienen los siguientes significados:

Tabla A. 2: Parámetros de las ACLs extendidas

Parámetro	Descripción
Número-de-lista-de-acceso	Identifica la lista utilizando un número de 100 a 199.
Permit | deny	Indica si esta entrada permite o bloquea la dirección especificada.
Protocolo	El protocolo, como *IP*, *TCP, UDP, ICMP, GRE* o *IGRE*
Source y destination	Identifica las direcciones de origen y de destino.
Máscara-de-origen y Máscara-de-destino	Máscara *wildcard*; los ceros indican posiciones que deben coincidir, mientras que los unos indican posiciones "sin importancia".
Operator operando	lt, gt, eq, neq (menor que, mayor que, igual, no igual) y número de puerto.
Established	Permite que pase el tráfico *TCP* si el paquete utiliza una conexión establecida (por ejemplo, si tiene establecidos los bits ACK).

Un ejemplo de cómo filtrar el tráfico mediante el uso de *ACL* extendidas podría ser el siguiente:

```
Router(config)# access-list 102 permit tcp host 192.168.2.5 host
192.168.1.254 eq ftp-data
Router(config)# access-list 102 permit tcp host 192.168.2.5 host
192.168.1.254 eq ftp
```

En este ejemplo definimos dos *ACL* extendidas que filtraran el tráfico *Telnet* y el tráfico *FTP* cuyo origen y destino sean 192.168.2.5 y 192.168.1.254 respectivamente.

Paso 2: Este paso se puede realizar de dos modos diferentes dependiendo de si se quiere aplicar la *ACL* directamente sobre la interfaz, o si se aplicará la *ACL* para clasificar el tráfico en clases. Este último caso es el que utilizamos en las configuraciones experimentales.

Configuración de la ACL sobre una interfaz: Se aplicará la *ACL* a una interfaz utilizando el comando access-group, como en este ejemplo:

```
Router(config-if)#{protocolo}access-group número-de-lista-de-acceso {in
| out}
```

Agrupa la ACL especificada con el número número-de-lista-de-acceso con la interfaz que se está configurando e indica si se aplicará a la entrada o a la salida de dicha interfaz.

Configuración de la ACL dentro de una clase: Como ya hemos comentado, se usa si queremos usar las *ACL* para filtrar paquetes y clasificarlos en clases de tráfico. No haremos el paso anterior. Después de definir la *ACL* mediante el paso 1, usaremos el comando **match access-**

group *numero-ACL* dentro de la configuración de la clase de tráfico. Esto hará que todos los paquetes filtrados por la *ACL* se consideren pertenecientes a la clase configurada. Por ejemplo:

```
Router(config)# access-list 1 permit 192.168.0.0 0.0.255.255
Router(config)# class-map match-all clase1
Router(config-cmap)# match access-group 1
Router(config-cmap)# exit
Router(config)# policy-map politica1
Router(config-pmap)# class clase1
Router(config-pmap-c)# police 8000 1000 1000 conform-action transmit
exceed-action set-qos-transmit 1 violate-action drop
Router(config-pmap-c)# exit
Router(config-pmap)# exit
Router(config)# interface fastethernet 0/0
Router(config-if)# service-policy output politica1
```

En el ejemplo anterior los pasos realizados son:

1. creamos una *ACL* que permita todos los paquetes *IP* de la red 192.168.0.0,
2. creamos una clase llamada clase 1,
3. usando el comando **match access-group** 1, todos los paquetes de la red 192.168.0.0 serán clasificados como pertenecientes a la clase1,
4. creamos una política llamada politica1,
5. realizamos un token-buket sobre el tráfico perteneciente a la clase1, es decir, el tráfico de la red 192.168.0.0,
6. y por último, aplicamos la política1 a la interfaz fastethernet 0/0.

Es importante señalar que en el caso de usar clases combinadas con las *ACL*, en vez de aplicar las *ACL* directamente sobre una interfaz. El tráfico que no cumpla con la *ACL* no será descartado, si no que será incluido como tráfico perteneciente a una clase por defecto *(class-default)*. Para más información dirigirse a la referencia [48].

Anexo B
Herramientas de Cisco para Realizar Traffic *Shaping*

B.1 Introducción

Cisco *IOS* soporta los siguientes métodos de *Traffic Shaping*:

- *Generic Traffic Shaping*
- *Frame Relay Traffic Shaping*
- *Class-Based Shaping* y *Distributed Class-Based Shaping*

Cisco *IOS QoS software* incluye dos tipos de *Traffic Shaping*: *Generic Traffic Shaping (GTS)* y *Frame Relay Traffic Shaping (FRTS)*. Ambos métodos son similares en implementación, aunque sus *command-line interfaces* se diferencian en algunos aspectos y usan diferentes tipos de colas para contener y dar forma al tráfico que retardan. En particular, el código subyacente que determina si hay suficiente crédito en el *token bucket* para que un paquete sea enviado o si el paquete se debe retardar es común para ambos mecanismos. Si un paquete se retarda, *GTS* usa un *WFQ* para mantener el tráfico retrasado. *FRTS* usa una *Custom Queue* o una *Priority Queue* para lo mismo, dependiendo de como se haya configurado.

También hay otro tipo de *Traffic Shaping:* el ***Class-Based Shaping*** que permite configurar *Generic Traffic Shaping (GTS)* sobre una clase de tráfico, especificando la tasa media o máxima del *Traffic Shaping*, y configurando políticas de encolamiento como *CBWFQ* (que se verá más adelante) dentro de *GTS* para las colas de salida.

Este último tipo de *Traffic Shaping* será el que se empleará dentro del *MQC* para configurar los Servicios Diferenciados.

El siguiente diagrama ilustra como una política de *QoS* divide el tráfico en clases y encola los paquetes que exceden las tasas de *Shaping* configuradas:

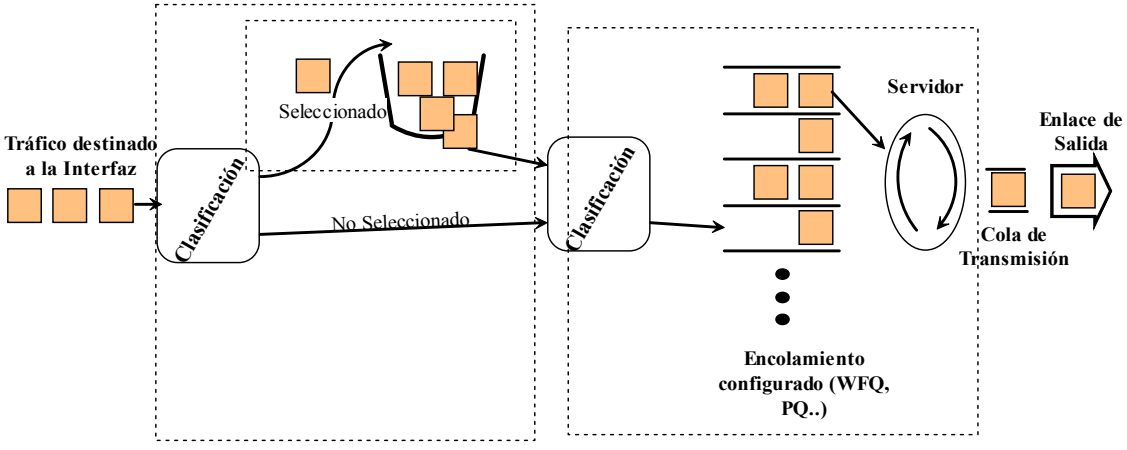

Figura B. 1: Funcionamiento de Generic Traffic *Shaping*

B.2 Generic Traffic *Shaping*

GTS acondiciona el tráfico reduciendo el flujo de tráfico de salida para evitar la congestión. Esto lo lleva a cabo reduciendo la tasa de tráfico usando el mecanismo *token bucket*.

GTS se aplica sobre las interfaces y puede usar *ACL* para seleccionar el tráfico a espaciar. Trabaja con varias tecnologías de la capa de enlace de datos, incluyendo *Frame Relay*, *ATM*, *Switched Multimegabit Data Service(SMDS)* y *Ethernet*.

B.3 *Class-Based Shaping*

Permite configurar *Generic Traffic Shaping (GTS)* sobre una clase de tráfico, especificando la tasa media o máxima del *traffic shaping*, y configurando *CBWFQ* dentro de *GTS*.

B.3.1 Configuración

Para configurar *Class-Based Shaping* utilizaremos primero los comandos **class-map** y **policy-map** para especificar el nombre de la clase y de la política, como hemos visto en ejemplos anteriores (ver *MQC*). Para especificar la media o tasa de pico emplearemos los siguientes comandos una vez que estamos configurando la política para esa clase de tráfico.

Tabla B. 1: Comandos de Configuración de *Class-Based Shaping*

	Comando	Propósito
Paso 1	`Router(config)# policy-map policy-map`	Especifica el nombre de la política que va a ser creada.
Paso 2	`Router(config-pmap)# class class-map-name`	Especifica el nombre de la clase sobre la que se aplica la política.
Paso 3	`Router(config-pmap-c)# shape {average | peak} cir [ bc] [ be]`	Especifica la tasa media o tasa de pico del *shaping*.
Paso 4	`Router(config-pmap-c)# shape max-buffers number-of-buffers`	(opcional) Especifica el número máximo de buffers permitidos en colas *shaping*.
Paso 5	`Router(config)# interface nombre-de-la-interfaz`	Especifica el nombre de la interfaz a la que se asociará la política.
Paso 6	`Router(config-if)# service-policy input|output policy-map`	Asignamos la política creada a la interfaz.

B.3.2 Ejemplos

Crear una política de QoS usando el comando shape

El siguiente ejemplo define una clase, c1. La clase c1 se configura para espaciar el tráfico a una tasa de 384 kbps con un tamaño normal de ráfaga de 15440 bits.

```
Router(config)# policy-map shape
Router(config-pmap)# class c1
Router(config-pmap-c)# shape average 38400 15440
Router(config)# interface Serial 3/3
Router(config-if)# service out shape
```

B.4 Diferencias entre *GTS* y *FRTS*

Como hemos mencionado, *GTS* y *FRTS* son similares en implementación, comparten las mismas estructuras de código y datos, pero difieren en sus *command-line interfaces* y los tipos de colas que usan.

Dos cosas en las que difieren son:

- *FRTS* soporta *shaping* sobre una base *per-DLCI*; *GTS* es configurable sobre una interfaz o subinterfaz.

- Para *GTS*, el tipo de cola es *WFQ*. Para *FRTS*, la cola puede ser: *WFQ* (configurada con el comando **frame-relay fair-queue**), una *strict priority queue* con *WFQ* (configurada con el comando **frame relay ip rtp priority** además del comando **frame-relay fair-queue**), *CQ, PQ* o *FIFO*.

Tabla con un resumen de las diferencias:

Tabla B. 2: Resumen de diferencias entre *GTS* y *FRTS*

	FRTS	***GTS***
Command-Line Interface	Clases de parámetros Aplica los parámetros a todos los circuitos virtuales de una interfaz mediante mecanismos de herencia. No soporta el comando **traffic group**	Aplica los parámetros por subinterfaz Soporta el comando **traffic group**
Colas soportadas	*WFQ*, strict priority queue con *WFQ*, *PQ*, *CQ* y *FIFO*.	*WFQ* por subinterfaz

Acrónimos

ABR Available Bit Rate
ACL Access Control List
AF Assured Forwarding
ATM Asynchronous Transfer Mode

BA Behavior Aggregate
Bc Committed Burst Size
Be Excess Burst Size

CAR Committed Access Rate
CBWFQ Class-Based Weighted Fair Queueing
CEF Cisco Express Forwarding
CIR Committed Information Rate
CLI Command Line Interface
CLP Cell Loss Priority
CoS Class of Service
CQ Custom Queueing
CU Currently Unused

DCE Data Communications Equipment
DE (Frame Relay) Discard Eligibility
DiffServ Differentiated Services
DLCI (Frame Relay) Data-Link Connection Identifier
DS Differentiated Services
DSCP Differentiated Services Code Point
DTE Data Terminal Equipment
DTS Defined Type of Service bits

ECN Explicit Congestion Notification
EEPROM Electrically Erasable Programmable Read-Only Memory
EF Expedited Forwarding

FIFO First In First Out
FQ Fair Queueing
FTP File Transfer Protocol

GPS Generalized Processor Sharing

HTTP HyperText Transfer Protocol

IETF Internet Engineering Task Force
IntServ Integrated Services
IOS Internetworking Operating System

MAC Medium Access Control
MQC Modular Quality of Service Command Line Interface
MPLS MultiProtocol Label Switching
MTU Maximum Transfer Unit

NVRAM Non-Volatile Random Access Memory

PHB Per-Hop Behavior
PQ Priority Queueing
PRI Primary Rate Interface
PVC Permanent Virtual Circuit

RDSI Red Digital de Servicios Integrados
RED Random Early Detection
RFC Request For Comments
ROM Read Only Memory
RSVP ReSerVation Protocol
RTP Real Time Protocol

SCFQ Self-Clocking Fair Queueing
SLA Service Level Agreement
SVC (ATM) Switched Virtual Circuit

TCA Traffic Conditioning Agreement
TCP Transport Control Protocol
TFTP Trivial File Transfer Protocol
ToS Type of Service

UDP User Datagram Protocol
UBR Unspecified Bit Rate

VBR Variable Bit Rate

WFQ Weighted Fair Queueing
WRED Weighted Random Early Detection

Referencias

[1] "*Quality of Service* Overview - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[2] Chuck Semeria y John W. Stewart III. White Paper: "Supporting Differentiated Service Classes in Large *IP* Networks", disponible en <www.juniper.net>

[3] Defense Advanced Research Projects Agency, "Internet Protocol, DARPA Internet Program Protocol Specification, RFC 791, Septiembre 1981.

[4] K. Nichols, S. Blake, F. Baker, D. Black, "Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers", RFC 2474, Diciembre 1998.

[5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss, "An Architecture for Differentiated Services", *RFC* 2475, Diciembre 1998

[6] P. Almquist "Type of Service in the Internet Protocol Suite", *RFC* 1349, Julio 1992

[7] D. Grossman, "New Terminology and Clarifications for Diffserv", *RFC* 3260, Abril 2002

[8] D. Black, S. Brim, B. Carpenter, F. Le Faucheur, "Per Hop Behavior Identification Codes" *RFC* 3140, Junio 2001

[9] J. Heinanen, F. Baker, W. Weiss, J. Wroclawski, "*Assured Forwarding PHB* Group", *RFC* 2597, Junio 1999

[10] B. Davie, A. Charny, JCR Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, D. Stiliadis, "An *Expedited Forwarding PHB*", *RFC* 3246, March 2002

[11] A. Charny, J.C.R. Bennett, K. Benson, J.Y. Le Boudec, A. Chiu, W. Courtney, S. Davari, V. Firoiu, C. Kalmanek, K.K.Ramakrishnan,"Supplemental Information for the New Definition of the EF *PHB* (*Expedited Forwarding* Per-Hop Behavior)", *RFC* 3247, Marzo 2002

[12] "Signalling Overview - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[13] "*DiffServ* - The Scalable End-to-End *QoS* Model", disponible en <www.cisco.com>

[14] Chuck Semeria, White Paper: "Supporting Differentiated Service Classes: Queue Scheduling Disciplines", disponible en <www.juniper.net>

[15] Chuck Semeria, White Paper: "Supporting Differentiated Service Classes: Active Queue Memory Management", disponible en <www.juniper.net>

[16] "Traffic *Policing* - Cisco *IOS* Release 12.2(2)T", disponible en <www.cisco.com>

[17] F, Baker, "Requirements for *IP* Version 4 Routers", *RFC* 1812, Junio 1995

[18] CISCO SYSTEMS, <www.cisco.com>

[19] "Using Cisco *IOS* Software - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[20] "Academia de Networking de Cisco Systems: Guía del Primer Año", 2ª Edición, Ed. Cisco Press, Capítulo 20, 2001

[21] "Academia de Networking de Cisco Systems: Guía del Primer Año", 2ª Edición, Ed. Cisco Press, Capítulo 21, 2001

[22] "Implementing *DiffServ* for End-to-End *Quality of Service* Overview - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[23] "Configuring the Modular *Quality of Service* Command-Line Interface - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[24] "Congestion Management Overview - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[25] "Congestion Avoidance Overview - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[26] "Classification Overview - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[27] "Configuring *Class-Based Packet Marking* - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[28] "Configuring *Class-Based Shaping* - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[29] "Configuring Weighted Fair Queueing - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[30] "Distributed Class-Based Weighted Fair Queueing and Distributed Weighted Random Early Detection - Release 12.1(5)T", disponible en <www.cisco.com>

[31] "Configuring Weighted Random Early Detection - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[32] "Low Latency Queueing - Cisco *IOS* Release 12.0(7)T", disponible en <www.cisco.com>

[33] "Cisco Express Forwarding (*CEF*) Release 11.1 CC", disponible en <www.cisco.com>

[34] "Comparing *Class-Based Policing* and Committed Access Rate", disponible en <www.cisco.com>

[35] "Configuring Committed Access Rate - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[36] "Cisco − Comparing Traffic *Policing* and Traffic *Shaping* for Bandwidth Limiting", disponible en <www.cisco.com>

[37] "Configuring Generic Traffic *Shaping* - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[38] "Configuring Frame Relay and Frame Relay Traffic *Shaping* - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[39] "*Policing* and *Shaping* Overview - Cisco *IOS Quality of Service* Solutions Configuration Guide", disponible en <www.cisco.com>

[40] "Cisco − Comparing the bandwidth and priority Commands of a *QoS* Service Policy", disponible en <www.cisco.com>

[41] Chuck Semeria, White Paper: "Supporting Differentiated Service Classes: *TCP* Congestion Control Mechanisms", disponible en <www.juniper.net>

[42] S. Floyd y V. Jacobson, "Random Early Detection gateways for Congestion Avoidance", IEEE/ACM Transactions on Networking, V.I N.4, p. 397-413, Agosto 1993.

[43] Hoja de Datos del Router multiservicio modular de la serie Cisco 2600, disponible en <www.cisco.com>

[44] Traffic Generator, disponible en <www.caip.rutgers.edu/~arni/linux/tg1.html>

[45] Netperf, disponible en <www.netperf.org>

[46] Programas generadores de tráfico: www.fokus.gmd.de/research/cc/glone/employees/sebastian.zander/private/trafficgen.html

[47] Maria Dolores Cano, Fernando Cerdan, Joan Garcia Haro, Josemaría Malgosa Sanahuja, "A New Proposal for Assuring Services in Internet", Proceedings of Internet Computing IC'02, CSREA Press, Vol. II, pp.379-384, Las Vegas, EEUU, Junio 2002

[48] "Academia de Networking de Cisco Systems: Guía del segundo Año", 2ª Edición, Ed. Cisco Press, Capítulo 6, 2001